AML Summary

Advanced Machine Learning Lecture at ETH

Lukas Diebold

January 16, 2026

Disclaimer

Help Improve These Notes

Feedback and contributions are welcome. If you spot mistakes, unclear passages, or missing intuition, please reach out or open an issue so the notes can be improved for everyone.

Contents

1 Math Preliminaries 7
1.1 Gibbs Distribution . . . 7
1.2 Conditional distribution of a multivariate Gaussian . . . 7
1.3 Schur complement . . . 7
1.4 Vector and matrix calculus . . . 8

2 Conceptual Foundation 9
2.1 What is Machine Learning? . . . 9
2.2 Conceptual foundation of inference . . . 9
2.3 Artificial Intelligence . . . 9
2.4 Extracting Value from Data - What is the problem? . . . 10
2.5 What does Generalization mean? . . . 10
2.6 Conceptional Foundation of Inference . . . 10

3 Fundamentals of Machine Learning 11
3.1 Efficiency: Cramér-Rao Bound . . . 11
3.2 Fisher information for n i.i.d. samples: . . . 13

4 Regression 14
4.1 Act 1: High-dimensional regression is unstable . . . 14
4.2 Act 2: Stability via Regularization . . . 15
4.3 Act 3: Polynomial regression via kernels . . . 16
4.4 Act 4: Neural Networks . . . 17
4.5 Some Things from the Slides . . . 18

5 Representations 20
5.1 Expected vs. empirical risk . . . 20
5.2 Comparing algorithms on shared test data . . . 20
5.3 Data, feature, and measurement spaces . . . 20
5.4 Scale types and transformation invariances . . . 21
5.5 Mathematical spaces underlying representations . . . 21
5.6 Probability spaces . . . 21

6 Gaussian Processes for Regression 23
6.1 From Bayesian linear regression to GPs . . . 23
6.2 What is a Gaussian process? . . . 23
6.3 Kernel design . . . 24
6.4 Prediction with Gaussian processes . . . 24
6.5 Validating kernels and hyperparameters . . . 25
6.6 Applications: control and fMRI . . . 25

7 Ensemble Methods 26
7.1 Why ensembles help . . . 26
7.2 Bagging and random forests . . . 27

7.3 Boosting . . . 27
7.4 Forward Stagewise Additive Modeling (FSAM) . . . 28
7.5 Gradient Descent Boosting . . . 28
7.6 AdaBoost . . . 29
7.7 Boosting as additive modeling . . . 29
7.8 Stacking . . . 30

8 Convex Optimization **31**
8.1 Convex sets, functions, and problems . . . 31
8.2 Illustrative example: closest point on a disk . . . 31
8.3 Standard convex programs in machine learning . . . 32
8.4 Lagrangian duality and KKT conditions . . . 32
8.5 Solving convex optimization problems . . . 33

9 Support Vector Machines **34**
9.1 Slater's Condition . . . 34
9.2 The Dual . . . 34
9.3 Kernelized SVMs . . . 35
9.4 Soft-margin SVMs . . . 35
9.5 Multiclass SVMs . . . 36
9.6 Structured SVMs . . . 36

10 Neural Networks **39**
10.1 Forward computation and parameterization . . . 39
10.2 Activation functions and intuition . . . 39
10.3 Loss, objective, and gradient descent . . . 39
10.4 The chain rule as the engine of backpropagation . . . 40
10.5 Backpropagation for a feed-forward network . . . 40

11 Transformers **42**
11.1 Tokenization and embeddings . . . 42
11.2 Self-attention: context for each token . . . 42
11.3 Attention as information retrieval . . . 42
11.4 Multi-head attention . . . 43
11.5 Cross-attention . . . 43
11.6 Masked self-attention for autoregressive decoding . . . 43
11.7 Positional encodings . . . 43
11.8 Residual connections and normalization . . . 43
11.9 The transformer block and full architecture . . . 44
11.10BERT as an encoder-only transformer . . . 44

12 Diffusion **45**

13 Graph Neural Networks **46**
13.1 Graphs, annotations, and learning objectives . . . 46

13.2 From layers to graph filters . . . 46
13.3 Message passing and expressive power . . . 46
13.4 Oversmoothing as a spectral phenomenon . . . 47
13.5 Mitigating oversmoothing . . . 47

14 Anomaly Detection 48
14.1 Anomalies . . . 48
14.2 Dimensionality Reduction . . . 48
14.2.1 First Stage . . . 48
14.2.2 General Case . . . 49
14.3 Fitting a GMM . . . 49
14.3.1 GMM Definition . . . 49
14.4 Fitting . . . 49
14.5 EM Algorithm . . . 50
14.6 Validation Metrics . . . 50
14.7 Combined Pipeline . . . 51

15 Reinforcement Learning 52
15.1 Agent-Environment Interaction Loop . . . 52
15.2 Markov Decision Processes . . . 52
15.3 Policies . . . 53
15.4 Trajectories and Returns . . . 53
15.5 The Markov Property . . . 53
15.6 Value Functions . . . 54
15.7 Bellman Equations . . . 54
15.7.1 Policy Evaluation . . . 54
15.7.2 Optimality . . . 55
15.8 Exploration versus Exploitation . . . 55
15.9 On-Policy versus Off-Policy Learning . . . 55
15.10Q-Learning . . . 56
15.10.1 The Update Rule . . . 56
15.11Deep Q-Networks (DQN) . . . 57
15.11.1 Function Approximation . . . 57
15.11.2 The DQN Algorithm . . . 57
15.12Policy-Based Methods and Policy Gradients . . . 58
15.12.1 Objective and Policy Gradient . . . 58
15.12.2 REINFORCE . . . 58
15.13Actor-Critic Methods . . . 59
15.13.1 Temporal-Difference Advantages . . . 59
15.13.2 Generalized Advantage Estimation (GAE) . . . 59
15.14Proximal Policy Optimization (PPO) . . . 59
15.15Entropy Regularization . . . 59
15.16Practical Training Notes . . . 60

16 Active Learning 61
16.1 Transductive information gain . . . 61
16.2 Information-based transductive learning . . . 61
16.3 Safe Bayesian optimization . . . 62
16.4 Safe Bayesian optimization via ITL . . . 62
16.5 Batch active learning . . . 62
16.6 Auxiliary definitions . . . 62
16.7 Coverage maximization and ProbCover . . . 63
16.8 Summary . . . 63

17 Counterfactual Invariance 64
17.1 Motivation: when correlations lie . . . 64
17.2 Shortcut learning and domain shifts . . . 64
17.3 Counterfactuals and invariance . . . 64
17.4 Two causal regimes . . . 65
17.5 Causal graphs and d-separation . . . 65
17.6 Confounding and selection pitfalls . . . 65
17.7 Necessary conditions for invariance . . . 66
17.8 Enforcing invariance via distribution matching . . . 66
17.9 Summary . . . 67

18 Variational Autoencoders 68
18.1 Representation learning without supervision . . . 68
18.2 The infomax principle and its limitations . . . 68
18.3 Latent-variable generative modeling . . . 68
18.4 A manifold perspective . . . 69
18.5 A Bayesian detour . . . 69

19 Non-parametric Bayesian Methods 70
19.1 Warm-up: Bayesian inference for a single Gaussian . . . 70
19.2 Multivariate Gaussians and conjugate priors . . . 70
19.3 Sampling with semi-conjugate priors . . . 71
19.4 Gibbs sampling in a nutshell . . . 71

20 Probably Approximately Correct Learning 73
20.1 From Empirical Patterns to Guarantees . . . 73
20.2 Instance, Concept, and Hypothesis Spaces . . . 73
20.3 The PAC Criterion . . . 73
20.4 Axis-Aligned Rectangles as a Running Example . . . 73
20.5 Induction Principles and Empirical Risk Minimization . . . 74
20.6 Uniform Convergence and the VC Inequality . . . 75
20.7 Finite Hypothesis Classes and Consistency . . . 75
20.8 Agnostic PAC Learning and Bayes Risk . . . 76
20.9 Controlling Complexity via VC Dimension . . . 76
20.10Empirical Risk Minimization for Hyperplanes . . . 77

20.11Strong and Weak Learning Perspectives 77

1 Math Preliminaries

1.1 Gibbs Distribution

The Gibbs distribution is a probability distribution that assigns likelihoods to states based on a cost function, with lower-cost states being more probable. Given a set of states $x \in \mathcal{X}$, a cost function $E(x)$, and an inverse temperature parameter $\beta > 0$, the Gibbs distribution is

$$p(x) = \frac{1}{Z} e^{-\beta E(x)}$$

where the partition function Z ensures normalization

$$Z = \sum_{x \in X} e^{-\beta E(x)}$$

1.2 Conditional distribution of a multivariate Gaussian

Let

$$\begin{bmatrix} a \\ b \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}, \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix} \right),$$

where Σ_{aa} and Σ_{bb} are covariance blocks and $\Sigma_{ab} = \Sigma_{ba}^{\top}$. Then the conditional distribution of a given b is again Gaussian with

$$\mathbb{E}[a \mid b] = \mu_a + \Sigma_{ab} \Sigma_{bb}^{-1} (b - \mu_b), \quad \mathrm{Cov}(a \mid b) = \Sigma_{aa} - \Sigma_{ab} \Sigma_{bb}^{-1} \Sigma_{ba}.$$

This identity underlies Gaussian process prediction, Bayesian linear regression posteriors, and Kalman filtering.

Derivation. Write the joint as a block Gaussian and use the Schur complement. The joint log-density (up to constants) is

$$\ell(a, b) = \tfrac{1}{2} \begin{bmatrix} a - \mu_a \\ b - \mu_b \end{bmatrix}^{\top} \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}^{-1} \begin{bmatrix} a - \mu_a \\ b - \mu_b \end{bmatrix}.$$

Completing the square in a (or applying the standard conditional Gaussian formula) yields

$$\mathbb{E}[a \mid b] = \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(b - \mu_b), \quad \mathrm{Cov}(a \mid b) = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}.$$

Equivalently, these follow from the block inversion identity and the Schur complement of Σ_{bb}.

1.3 Schur complement

For a block matrix $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ with D invertible, the **Schur complement of D in M** is

$$S = A - BD^{-1}C.$$

Key identities (when the required inverses exist):

- Determinant: $\det(M) = \det(D)\, \det(S)$.
- Block inverse: $M^{-1} = \begin{bmatrix} S^{-1} & -S^{-1}BD^{-1} \\ -D^{-1}CS^{-1} & D^{-1} + D^{-1}CS^{-1}BD^{-1} \end{bmatrix}$.
- If M is symmetric positive (semi)definite and D is invertible, then S is also positive (semi)definite.

Symmetrically, if A is invertible, the Schur complement of A is $D - CA^{-1}B$. In Gaussian conditioning, $\mathrm{Cov}(a \mid b)$ equals the Schur complement of Σ_{bb} in the joint covariance.

1.4 Vector and matrix calculus

We use the convention that gradients are column vectors. For $x \in \mathbb{R}^d$ and matrices/vectors of compatible dimensions, the following identities are used repeatedly:

- $\nabla_x(b^\top x) = b$ and $\nabla_x(x^\top b) = b$.
- $\nabla_x(x^\top Ax) = (A + A^\top)x$ (and $= 2Ax$ if A is symmetric).
- $\nabla_x\|x\|_2^2 = \nabla_x(x^\top x) = 2x$.
- $\nabla_\beta\|y - X\beta\|_2^2 = -2X^\top(y - X\beta)$.

Derivation. For $f(x) = b^\top x = \sum_i b_i x_i$, we have $\partial f/\partial x_i = b_i$, so $\nabla_x f = b$. The same holds for $x^\top b$ since it is the same scalar.

Derivation. To see why $\nabla_x(x^\top Ax) = (A + A^\top)x$, expand component-wise:

$$f(x) = x^\top Ax = \sum_{i,j} x_i A_{ij} x_j.$$

Then the k-th component of the gradient is

$$\frac{\partial f}{\partial x_k} = \sum_j A_{kj}x_j + \sum_i x_i A_{ik} = (Ax)_k + (A^\top x)_k.$$

Stacking these components yields $\nabla_x f = (A + A^\top)x$.

Derivation. For $f(x) = \|x\|_2^2 = x^\top x$, apply the rule $\nabla_x(x^\top Ax) = (A + A^\top)x$ with $A = I$. Since I is symmetric, $\nabla_x f = 2Ix = 2x$.

Derivation. For $g(\beta) = \|y - X\beta\|^2 = (y - X\beta)^\top(y - X\beta)$, expand to $y^\top y - 2y^\top X\beta + \beta^\top X^\top X\beta$ and apply the previous rules:

$$\nabla_\beta g = -2X^\top y + 2X^\top X\beta.$$

Here we use that $y^\top X\beta = (X^\top y)^\top\beta$, so $\nabla_\beta(-2y^\top X\beta) = -2X^\top y$.

Rewriting yields $\nabla_\beta g = -2X^\top(y - X\beta)$.

2 Conceptual Foundation

2.1 What is Machine Learning?

"ML is a mathematization of epistemolagy!". In philosophy, it is the science of knowledge, the science of what can be known. This is relevant, because in ML we are interested in systems that produce/generate knowledge.

The goal is then to observe 'reality' and draw conclusions from the observations. This can be seen as a perception-action cycle. Where perception is the result of our observations (typically in a data space $\mathcal{X}$) and the actions are part of a hypothesis space. To go from the data space to the hypothesis class $\mathcal{C}$ we generally use an algorithm A. See Figure 1 for an overview.

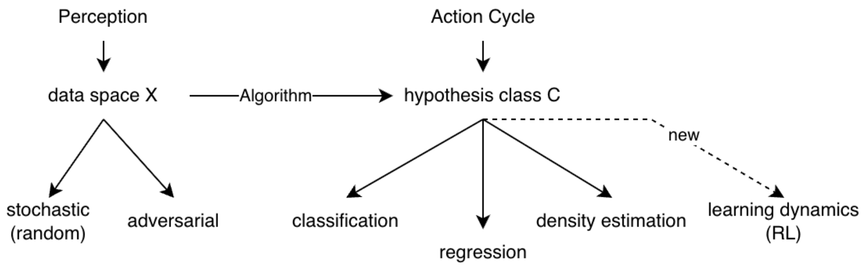


Figure 1: Overview ML

Information processing occurs when $|\mathcal{X}| \gg |\mathcal{C}|$. Taking the example of combinatorial optimization problems, $|\mathcal{X}|$ would be the space of weighted graphs, and we look for a color of the graph or something similar. Then $|\mathcal{X}| \approx K^{\binom{n}{2}}$ and $|\mathcal{C}| \approx e^{n \log n}$ so we obserge that $|\mathcal{X}|$ is much larger.

2.2 Conceptual foundation of inference

1. Perception of realits is mediated by data of senses/sensors
2. Data are stochastic $\rightarrow$ porbabilistic
3. Sensing restricts us to selected aspects of reality
4. Humans interpret data by a huge reduction in degrees of freedom
   $|x| \gg |e|$ (space of graphs $\gg$ space of colorings, cycles, spanning trees)
5. Tuple ({data }, {hypotheses }) define models:

$$\begin{aligned} \mathcal{A} : \mathcal{X} &\longrightarrow \mathcal{C} \\ x &\longmapsto c = \mathcal{A}(x) \end{aligned}$$

6. Experiments ϵ provide us with data
7. Learning means interpreting X w.r.t hypotheses $\mathcal{C}$

2.3 Artificial Intelligence

Taking a high level view. We live in very high dimensional data, which we are not able to fully process. We use algorithms to make sense of the data, and this then informs our values, from this we get the following relation

$$\text{Data} \longrightarrow \text{ Algorithms } \mathcal{A} \longrightarrow \text{ Values}$$

In epistemology, we differentiate between **deduction** and **induction**. Deduction is a form of reasoning in which the conclusion follows necessarily from the premises, while induction tries to generalize, that is, the conclusion goes beyond the premises and generally probabilistic. We can construct a model in which deduction and induction form a **feedback loop**, not two isolated methods. In simpler terms, on one side we try to formulate axioms from our observations (empirical data), and on the other side we then use these axioms and derive logical consequences from them. This is not anything new, and this cycle generally informs the model of "Theory, Experiment, Computation" in science. What has changed with ML/AI is that we're now in the era of non-parametric modeling.

2.4 Extracting Value from Data - What is the problem?

- Algorithms that process inputs with noise compute random variables as outputs!
- Algorithms should compute typical solutions!
- When do algorithms generalize over noise/model mismatch?
- How can algorithms autonomously improve performance?

2.5 What does Generalization mean?

- Out-of-sample risk

$$\theta^\star\left(X'\right) \sim \mathbb{P}^{\mathcal{A}}(\theta \mid X) \in \arg\min_{\mathbb{P}(.|.)} \mathbb{E}_{X'}\mathbb{E}_{\theta|X'}\mathbb{E}_{X''|X'}R\left(\theta, X''\right)$$

  where X' is the training data and X'' is the test data, so this is the risk where θ is conditioned on the training data (trained the model). This is the standard model.

- Log loss of posterior (risks and probabilities are dependent!)

$$\begin{aligned}\theta^\star\left(X'\right) \sim \mathbb{P}^{\mathcal{A}}(\theta \mid X) &\in \arg\min_{\mathbb{P}(.|.)} \mathbb{E}_{X'}\mathbb{E}_{\theta|X'}\mathbb{E}_{X''|X'}\left(-\log \mathbb{P}\left(\theta \mid X''\right)\right)\\ &\in \arg\min_{\mathbb{P}(.|.)} \mathbb{E}_{X'}\mathbb{E}_{\theta|X'}\mathbb{E}_{X''|X'}\left(\beta R\left(\theta, X''\right) + \log Z\right)\end{aligned}$$

  This objective scores the full posterior: on average it should concentrate on parameters that make future data likely. Using the Gibbs form $\mathbb{P}(\theta \mid X'') \propto \exp(-\beta R(\theta, X''))$, the log loss becomes $\beta R(\theta, X'') + \log Z$, so minimizing expected log loss matches minimizing expected risk up to the normalizer. Unlike the first criterion, which evaluates the risk of a single parameter draw, this one emphasizes posterior calibration via how probability mass is distributed.

- Posterior agreement

$$\theta^\star\left(X'\right) \sim \mathbb{P}^{\mathcal{A}}(\theta \mid X) \in \arg\min_{\mathbb{P}(.|.)} \mathbb{E}_{X'}\mathbb{E}_{X''|X'}\left(-\log \mathbb{E}_{\theta|X'}\mathbb{P}\left(\theta \mid X''\right)\right)$$

  This criterion maximizes the average overlap between the posterior from training data X' and the posterior induced by future data X'': $\mathbb{E}_{\theta|X'}\mathbb{P}(\theta \mid X'')$ is a similarity score, and the outer $-\log$ turns it into a loss. Unlike the second objective, which takes the expectation of a log loss for a sampled θ, here the log is outside the expectation over θ, so it rewards overall posterior agreement rather than pointwise posterior accuracy.

2.6 Conceptional Foundation of Inference

Our perception of reality is mediated by data from our senses or sensors, and those data are shaped by chance. Creatures interpret selected aspects of reality through hypotheses in order to survive and reproduce, and data together with hypotheses define models that enable judgments, decisions, and actions. In AI/ML, algorithms formalize the relation between data and hypotheses, for example by selecting models.

3 Fundamentals of Machine Learning

Machine learning is about inferring models from data. We begin with Bayes' rule to show how likelihood and prior combine to form the posterior, which is the full probabilistic description of what we know after seeing data:

$$\mathbb{P}(\text{model} \mid \text{data}) = \frac{\mathbb{P}(\text{ data } \mid \text{ model })\mathbb{P}(\text{ model })}{\mathbb{P}(\text{ data })}$$

In many practical settings, we compress the posterior into a single point estimate. A common choice is the **maximum likelihood (ML)** approach, which selects the model that maximizes the likelihood of observing the data:

$$\widehat{\text{model}} \in \arg\max_{\text{model}} \mathbb{P}(\text{data} \mid \text{model})$$

Under regularity conditions, the ML estimator $\widehat{\text{model}}_n$ is consistent, asymptotically normal, and asymptotically efficient. This highlights why ML remains a standard baseline: it ignores the prior but becomes reliable as data grow.

To understand what makes an estimator "good," we introduce two core properties that we will use throughout.

Consistency: A point estimator $\hat{\theta}_n$ of the parameter $\theta = \theta_0$ is consistent if it converges in probability to the true parameter:

$$\forall \varepsilon > 0, \mathbb{P}\left(\left|\hat{\theta}_n - \theta_0\right| > \varepsilon\right) \xrightarrow{n\to\infty} 0$$

More formally, using the ε-δ definition:

$$\forall \theta, \forall \varepsilon, \delta > 0, \exists n_0, \forall n > n_0, \mathbb{P}\left(\left|\widehat{\theta}_n - \theta\right| < \varepsilon\right) > 1 - \delta$$

Efficiency: An estimator $\hat{\theta}_n$ is efficient if it achieves the minimum mean squared error among all estimators:

$$\hat{\theta}_n = \arg\min_{\hat{\theta}} \mathbb{E}\left[\left(\hat{\theta}_n - \theta_0\right)^2\right]$$

Consistency is a long-run guarantee, while efficiency quantifies finite-sample precision.

This raises the practical question of precision: how well can we estimate θ from n samples? The Cramér-Rao bound provides a fundamental lower bound on the variance of any unbiased estimator.

3.1 Efficiency: Cramér-Rao Bound

Problem: What is the best achievable precision for parameter estimation, given a likelihood model? We want a benchmark that applies to any estimator so we can judge how close a procedure gets to optimal performance.

Given a likelihood $p(y \mid \theta)$ for $\theta \in \Theta$ and data $y_1, \ldots, y_n \sim p(y \mid \theta = \theta_0)$, we ask: How precisely can we estimate $\theta = \theta_0$ given n samples?

For an estimator $\hat{\theta}(y_1, \ldots, y_n)$, we measure precision via the mean squared error:

$$\mathbb{E}_{y|\theta}\left[(\hat{\theta} - \theta)^2\right]$$

The Cramér-Rao bound (Equation 1) shows that this error cannot be made arbitrarily small; it is constrained by the information in the data and by estimator bias.

$$\mathbb{E}_{y|\theta}\left[(\hat{\theta} - \theta)^2\right] \geq \frac{\left(\frac{\partial}{\partial\theta} b_{\hat{\theta}} + 1\right)^2}{\mathbb{E}_{y|\theta}\left[\Lambda^2\right]} + b_{\hat{\theta}}^2 \tag{1}$$

Estimation error is fundamentally limited by the curvature of the likelihood (via the score variance) and the estimator bias. Even the best estimator cannot beat this limit for a given model.

Derivation. The derivation relies on the score and the estimator bias. The score measures local sensitivity of the log-likelihood to θ, while the bias captures systematic estimation error.

- Score: $\Lambda = \frac{\partial}{\partial\theta} \log p(y \mid \theta) = \frac{\frac{\partial}{\partial\theta} p(y|\theta)}{p(y|\theta)}$
- Bias: $b_{\hat{\theta}} = \mathbb{E}_{y|\theta}\left[\hat{\theta}\left(y_1, \ldots, y_n\right)\right] - \theta$

We will relate the score to the estimator, express their covariance in terms of bias, and then use Cauchy-Schwarz to obtain a limit on the mean squared error.

Expected score: The score has zero mean. This follows from the normalization of $p(y \mid \theta)$ and ensures the score behaves like a centered random variable.

$$\begin{aligned}\mathbb{E}_{y|\theta}[\Lambda] &= \int p(y \mid \theta) \frac{\frac{\partial}{\partial\theta} p(y \mid \theta)}{p(y \mid \theta)} dy \\ &= \frac{\partial}{\partial\theta} \int p(y \mid \theta) dy = \frac{\partial}{\partial\theta} 1 = 0\end{aligned}$$

Score-estimator product:

$$\begin{aligned}\mathbb{E}_{y|\theta}[\Lambda\hat{\theta}] &= \int p(y \mid \theta) \frac{\frac{\partial}{\partial\theta} p(y \mid \theta)}{p(y \mid \theta)} \hat{\theta} dy \\ &= \frac{\partial}{\partial\theta}\left(\int p(y \mid \theta)\hat{\theta} dy\right) \\ &= \frac{\partial}{\partial\theta}\left(\mathbb{E}_{y|\theta}\hat{\theta}\right) = \frac{\partial}{\partial\theta}\left(b_{\hat{\theta}} + \theta\right) = \frac{\partial}{\partial\theta} b_{\hat{\theta}} + 1\end{aligned}$$

This identity ties the score to the bias derivative and will connect estimation error to likelihood curvature.

Cross-correlation:

$$\mathbb{E}_{y|\theta}\left[(\Lambda - \mathbb{E}\Lambda)\left(\hat{\theta} - \mathbb{E}\hat{\theta}\right)\right] = \mathbb{E}_{y|\theta}[\Lambda\hat{\theta}] - \mathbb{E}_{y|\theta}[\Lambda]\mathbb{E}\hat{\theta} = \mathbb{E}_{y|\theta}[\Lambda\hat{\theta}]$$

since $\mathbb{E}[\Lambda] = 0$. This expresses the covariance between the score and the estimator in terms of the score-estimator product.

Cauchy-Schwarz inequality: Applying Cauchy-Schwarz to the cross-correlation:

$$\left(\mathbb{E}_{y|\theta}[\Lambda(\hat{\theta} - \mathbb{E}\hat{\theta})]\right)^2 \leq \mathbb{E}_{y|\theta}\left[\Lambda^2\right] \mathbb{E}_{y|\theta}\left[(\hat{\theta} - \mathbb{E}\hat{\theta})^2\right]$$

Expanding the right-hand side:

$$\begin{aligned}\mathbb{E}_{y|\theta}\left[(\hat{\theta} - \mathbb{E}\hat{\theta})^2\right] &= \mathbb{E}_{y|\theta}\left[(\hat{\theta} - \theta + \theta - \mathbb{E}\hat{\theta})^2\right] \\ &= \mathbb{E}_{y|\theta}\left[(\hat{\theta} - \theta)^2\right] - b_{\hat{\theta}}^2\end{aligned}$$

Therefore, bounding the covariance by the product of variances yields a lower bound on the mean squared error once we substitute the bias term:

$$\left(\frac{\partial}{\partial\theta} b_{\hat{\theta}} + 1\right)^2 \leq \mathbb{E}_{y|\theta}\left[\Lambda^2\right]\left(\mathbb{E}_{y|\theta}\left[(\hat{\theta} - \theta)^2\right] - b_{\hat{\theta}}^2\right)$$

Rearranging yields the **Cramér-Rao bound**:

$$\mathbb{E}_{y|\theta}\left[(\hat{\theta} - \theta)^2\right] \geq \frac{\left(\frac{\partial}{\partial\theta} b_{\hat{\theta}} + 1\right)^2}{\mathbb{E}_{y|\theta}\left[\Lambda^2\right]} + b_{\hat{\theta}}^2$$

Fisher information: The expected squared score is called the Fisher information:

$$I(\theta) := \mathbb{E}_{y|\theta}\left[\Lambda^2\right] = \int p(y \mid \theta)\left(\frac{\partial}{\partial\theta}\log p(y \mid \theta)\right)^2 dy$$

It measures how much information the data contains about the parameter θ. Higher Fisher information means we can estimate θ more precisely, which emphasizes that precision is controlled by data informativeness, not just by the estimator.

Remarks:

- For unbiased estimators ($b_{\hat{\theta}} = 0$), the bound simplifies to $\mathbb{E}[(\hat{\theta} - \theta)^2] \geq 1/I(\theta)$.
- The bound reveals a trade-off for biased estimators: reducing bias derivative $\frac{\partial}{\partial\theta} b_{\hat{\theta}}$ vs. reducing squared bias $b_{\hat{\theta}}^2$. Unbiased estimators are not always optimal!

3.2 Fisher information for n i.i.d. samples:

The Fisher information of n i.i.d. random variables is n times the Fisher information of a single random variable. This shows that precision improves linearly with sample size.

Derivation.

$$\begin{aligned}
I^{(n)}(\theta) &= \mathbb{E}_{y_1,\ldots,y_n|\theta}\left[\Lambda^2\right] \\
&= \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log p\left(y_1,\ldots,y_n \mid \theta\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{n}\frac{\partial}{\partial\theta}\log p\left(y_i \mid \theta\right)\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{n}\Lambda_i\right)^2\right] \\
&= \sum_{i=1}^{n}\mathbb{E}\left[\Lambda_i^2\right] + \sum_{i\neq j}\mathbb{E}\left[\Lambda_i\right]\mathbb{E}\left[\Lambda_j\right] \\
&= \sum_{i=1}^{n}\mathbb{E}\left[\Lambda_i^2\right] \\
&= nI(\theta)
\end{aligned}$$

where the cross-terms vanish because $\mathbb{E}[\Lambda_i] = 0$ and the samples are independent.

4 Regression

4.1 Act 1: High-dimensional regression is unstable

We assume X and y are distributed according to a distribution p_* (i.e. $X, y \sim p_*$), where the output follows a noisy linear model:

$$y = f_*(x) + \varepsilon \quad \text{with } \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

Here f_* is the true (unknown) regression function and ε is additive Gaussian noise with variance σ^2. Our task is to estimate f_* from training data $D = \{(x_i, y_i)\}_{i=1}^n \sim p_*$.

The problem in this form is not tractable because the space of all possible functions is too large. We therefore restrict ourselves to linear functions:

$$f_*(x) = \beta^\top x$$

where $\beta \in \mathbb{R}^d$ is a parameter vector. Given the Gaussian noise assumption, each observation has likelihood $p(y_i|x_i, \beta) = \mathcal{N}(\beta^\top x_i, \sigma^2)$. We solve for β using Maximum Likelihood Estimation (MLE):

$$\begin{aligned}
\widehat{\beta} &= \arg\max_{\beta \in \mathbb{R}^d} p(D \mid \beta) \\
&= \arg\max_{\beta} \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \beta^\top x_i)^2}{2\sigma^2}\right) \\
&= \arg\min_{\beta} \sum_{i=1}^n (y_i - \beta^\top x_i)^2 \\
&= \arg\min_{\beta} \operatorname{MSE}(D, \beta)
\end{aligned}$$

Maximizing the log-likelihood is equivalent to minimizing the mean squared error (MSE). The closed-form solution depends on whether we have more features than samples or vice versa:

$$\begin{aligned}
\widehat{\beta} &= (X^\top X)^{-1} X^\top y \quad \text{(when } d < n\text{, more samples than features)} \\
&= X^\top (X X^\top)^{-1} y \quad \text{(when } d > n\text{, more features than samples)}
\end{aligned}$$

These are algebraically equivalent by the Woodbury matrix identity. The first formula is the standard *ordinary least squares (OLS)* estimator, where

$$X = \begin{bmatrix} -x_1- \\ \vdots \\ -x_n- \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

This estimator has some interesting properties. It is unbiased and, by the Gauss-Markov Theorem, it is the best linear unbiased estimator, i.e. it attains the smallest variance among all linear unbiased estimators. Thus, from the formula

$$\text{error} = \text{bias}^2 + \text{variance} + \text{noise}$$

we find that this estimator is the one with the smallest error of all the unbiased estimators. Then why does no-one use this estimator? If we introduce a bit of bias, we can significantly reduce the variance.

To understand the instability, we analyze $\operatorname{Var}(\hat{\beta})$ using the singular value decomposition (SVD) $X = UDV^\top$, where $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{d \times d}$ are orthogonal, and D is diagonal with singular values $D_{11} \geq D_{22} \geq \ldots \geq 0$. Plugging this into the OLS formula:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y = (V D^\top U^\top U D V^\top)^{-1} V D^\top U^\top y = V D^{-1} U^\top y$$

Since $y = X\beta_* + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$, and we multiply y by the deterministic matrix $VD^{-1}U^\top$, the estimator $\hat{\beta}$ is also Gaussian. Its variance is:

$$\operatorname{Var}(\hat{\beta}) = \operatorname{Var}(VD^{-1}U^\top y) = VD^{-1}U^\top \operatorname{Var}(y) U D^{-1} V^\top = \sigma^2 V D^{-2} V^\top = \sigma^2 \sum_{i \leq r} \frac{1}{D_{ii}^2} V_i V_i^\top$$

where $r = \text{rank}(X)$ and V_i is the i-th column of V (the i-th right singular vector).

The problem: In high-dimensional data, features are often correlated (e.g., pixel intensities in images, gene expressions). This makes X close to low-rank, so several singular values D_{ii} are very small. The variance contributions $1/D_{ii}^2$ then explode for these directions, causing massive instability in $\hat{\beta}$ even though it remains unbiased. Small noise in y gets amplified enormously in directions with small singular values, leading to wild predictions on test data.

4.2 Act 2: Stability via Regularization

The solution to the variance blow-up is to introduce regularization, which adds a small amount of bias in exchange for a large reduction in variance. We can derive regularization naturally from a Bayesian perspective.

The typical process of **Bayesian inference** goes through the following stages:

1. Prior $\beta \sim \mathcal{N}(0, \tau^2 I)$ — we assume β is drawn from a Gaussian centered at zero with variance τ^2. This encodes our belief that coefficients should not be too large.
2. Likelihood $p(D|\beta) = \prod_i \mathcal{N}(y_i|\beta^\top x_i, \sigma^2)$ — same Gaussian noise model as before.
3. Posterior (via Bayes' rule) $p(\beta|D) \propto p(\beta)p(D|\beta) \propto \exp\left(-\frac{1}{2\sigma^2}\text{MSE}(D,\beta) - \frac{1}{2\tau^2}\|\beta\|^2\right)$

The posterior combines the likelihood (fit to data) with the prior (regularization). To derive the optimization objective, we use the fact that maximizing the posterior probability is equivalent to minimizing its negative logarithm. From Bayes' rule:

$$p(\beta|D) \propto p(\beta)p(D|\beta) = \mathcal{N}(0, \tau^2 I) \cdot \prod_i \mathcal{N}(y_i|\beta^\top x_i, \sigma^2)$$

Taking the negative log (up to constant terms):

$$-\log p(\beta|D) \propto \underbrace{-\log p(\beta)}_{-\frac{1}{2\tau^2}\|\beta\|^2+\text{const}} + \underbrace{-\log p(D|\beta)}_{-\sum_i \log \mathcal{N}(y_i|\beta^\top x_i, \sigma^2)}$$

For Gaussian distributions, $-\log \mathcal{N}(y|\mu, \sigma^2) = \frac{(y-\mu)^2}{2\sigma^2} + \text{const}$, so:

$$-\log p(\beta|D) \propto \frac{1}{2\tau^2}\|\beta\|^2 + \frac{1}{2\sigma^2}\sum_i (y_i - \beta^\top x_i)^2$$

Minimizing this gives the MAP (maximum a posteriori) estimate:

$$\begin{aligned}\hat{\beta}_{\text{MAP}} &= \arg\min_\beta \left[\frac{1}{2\sigma^2}\sum_i (y_i - \beta^\top x_i)^2 + \frac{1}{2\tau^2}\|\beta\|^2\right] \\ &= \arg\min_\beta \left[\sum_i \left(y_i - \beta^\top x_i\right)^2 + \lambda\|\beta\|^2\right]\end{aligned}$$

This is precisely **ridge regression** with regularization parameter $\lambda = \sigma^2/\tau^2$. The ℓ^2 penalty $\|\beta\|^2$ shrinks coefficients toward zero. If we instead use a Laplace prior $p(\beta) \propto \exp(-|\beta|/\tau)$ with heavier tails, we obtain **lasso regression** with an ℓ^1 penalty $\|\beta\|_1$, which promotes sparsity.

The prior variance τ^2 controls the bias-variance trade-off: small τ^2 (big λ) means strong regularization (more bias, less variance), while large τ^2 (small λ) recovers OLS (no bias, high variance). The MAP (maximum a posteriori) solution is:

$$\hat{\beta}_{\text{MAP}} = \left(X^\top X + \lambda I\right)^{-1} X^\top y \tag{2}$$

Derivation. Start from the MAP/ridge objective in matrix form

$$J(\beta) = \sum_i (y_i - x_i^\top \beta)^2 + \lambda\,\|\beta\|^2 \;=\; \|y - X\beta\|^2 + \lambda\,\beta^\top\beta.$$

Differentiate and set the gradient to zero:

$$\begin{aligned}\nabla_\beta J(\beta) &= -2X^\top(y - X\beta) + 2\lambda\,\beta \\ &= 2\,(X^\top X + \lambda I)\beta - 2X^\top y \;=\; 0.\end{aligned}$$

Thus the normal equations are $(X^\top X + \lambda I)\,\hat\beta = X^\top y$. For $\lambda > 0$, the matrix $X^\top X + \lambda I$ is positive definite and hence invertible, which yields the closed form in (2).

Compare this to OLS: the regularization term λI is added to $X^\top X$ before inversion, preventing ill-conditioning. Using SVD again to analyze the variance:

$$\mathrm{Var}(\hat\beta_{\mathrm{MAP}}\ ) = \sigma^2 \sum_{i\le r} \frac{D_{ii}^2}{\left(D_{ii}^2 + \lambda\right)^2} V_i V_i^\top \tag{3}$$

Derivation. Using $\hat\beta_{\mathrm{MAP}} = (X^\top X + \lambda I)^{-1}X^\top y$ and $y = X\beta_* + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$, the estimator is affine in y, so its variance depends only on the noise:

$$\mathrm{Var}(\hat\beta_{\mathrm{MAP}}) = (X^\top X + \lambda I)^{-1}X^\top\,\mathrm{Var}(y)\,X(X^\top X + \lambda I)^{-1} = \sigma^2(X^\top X + \lambda I)^{-1}X^\top X(X^\top X + \lambda I)^{-1}.$$

With the SVD $X = UDV^\top$, we have $X^\top X = VD^2V^\top$ and $(X^\top X + \lambda I)^{-1} = V(D^2 + \lambda I)^{-1}V^\top$. Substituting,

$$\mathrm{Var}(\hat\beta_{\mathrm{MAP}}) = \sigma^2 V(D^2 + \lambda I)^{-1}D^2(D^2 + \lambda I)^{-1}V^\top = \sigma^2 \sum_{i\le r} \frac{D_{ii}^2}{(D_{ii}^2 + \lambda)^2} V_i V_i^\top,$$

which yields (3).

The key is the **shrinkage factor** $\frac{D_{ii}^2}{(D_{ii}^2 + \sigma^2/\tau^2)^2}$. For large singular values ($D_{ii}^2 \gg \sigma^2/\tau^2$), this is close to $1/D_{ii}^2$ (like OLS). For small singular values ($D_{ii}^2 \ll \sigma^2/\tau^2$), the factor is approximately $\tau^4 D_{ii}^2/\sigma^4$, which decays much more slowly than $1/D_{ii}^2$. This prevents variance blow-up in the problematic low-variance directions, stabilizing the estimator at the cost of introducing bias (shrinking coefficients toward zero).

4.3 Act 3: Polynomial regression via kernels

Now we change our assumption for $f_*(x)$ to allow for nonlinear functions. We model f_* as a linear function in an infinite-dimensional feature space:

$$f_*(x) = \varphi(x)^\top \beta_*$$

where $\beta_* \in \mathbb{R}^\infty$ and $\varphi(x)$ maps each input to an infinite-dimensional polynomial feature representation:

$$\varphi(X) = K_x \left(\frac{x_1^{\alpha_1} \ldots x_d^{\alpha_d}}{\sqrt{\alpha_1! \ldots \alpha_d!}}\right)_{\alpha \in \mathbb{N}^d}$$

This includes all polynomial terms of all degrees. The normalization by factorials ensures the inner product has a clean closed form.

Remarkably, the inner product of two infinite-dimensional feature vectors yields the radial basis function (RBF) kernel. For $x, x' \in \mathbb{R}^a$:

$$\begin{aligned}\varphi(x)^\top \varphi\left(x'\right) &= K_{RBF}\left(x, x'\right) \\ &= \exp\left(-\frac{1}{2}||x - x'||^2\right)\end{aligned}$$

This follows from the Taylor expansion of the exponential function. The RBF kernel measures similarity: it is 1 when $x = x'$ and decays as points move apart.

We still want to minimize the MSE, but now in the infinite-dimensional feature space:

$$\begin{aligned}\hat\beta &= \arg\min_{\beta \in \mathbb{R}^\infty} \frac{1}{n} \sum_{i\le n} \left(y_i - \varphi\left(x_i\right)^\top \beta\right)^2 \\ &= \Phi^\top \left(\Phi\Phi^\top\right)^{-1} y\end{aligned}$$

where

$$\Phi = \begin{bmatrix} \varphi(x)^\top \\ \varphi(x_2)^\top \\ \varphi(x_n)^\top \end{bmatrix} \in \mathbb{R}^{n\times\infty}$$

Despite β living in infinite dimensions, the representer theorem guarantees the solution lies in the span of the training features, so we can work with the $n \times n$ Gram matrix $\Phi\Phi^\top$ instead of the infinite-dimensional feature space directly.

To make a prediction at test point x_*, we compute:

$$\begin{aligned} \hat{y}_* &= \varphi(x_*)^\top \hat{\beta} \\ &= \varphi(x_*)^\top \Phi^\top \left(\Phi\Phi^\top\right)^{-1} y \\ &= k(x_*)^\top K^{-1} y \end{aligned}$$

where $k(x_*) = \left(\varphi(x_*)^\top \varphi(x_i)\right)_{1\le i\le n} = (K_{RBF}(x_*, x_i))_{1\le i\le n}$ is an n-dimensional vector of kernel evaluations between the test point and each training point, and $K_{ij} = \varphi(x_i)^\top\varphi(x_j) = K_{RBF}(x_i, x_j)$ is the $n \times n$ kernel matrix.

This is the **kernel trick**: we never explicitly construct the infinite-dimensional $\varphi(\cdot)$. Instead, we only compute inner products via the kernel function K_{RBF}, which can be evaluated in closed form. The prediction is a weighted combination of training outputs, where the weights depend on how similar the test point is to each training point.

The problem is that the inversion of the matrix is $O(n^3)$, which becomes costly for large datasets even though we avoided the infinite feature map explicitly.

4.4 Act 4: Neural Networks

We assume f_* has only a single, very wide hidden layer.

$$f_*(X) = \frac{1}{\sqrt{m}} \sum_{i\le m} \alpha_i \phi\left(\omega_i^\top X\right)$$

where ϕ is a nonlinear activation function (e.g. ReLU, tanh), and the network has m hidden units. The parameters are $\theta = \{\alpha_i, w_i\}_{i\le m}$, i.e. both the output weights α_i and the input weights w_i are learned. We initialize with

$$\theta_0 \sim \mathcal{N}\left(0, w^2\right)$$

and we update our parameters using gradient descent.

$$\theta_{t+1} \leftarrow \theta_t - \eta\nabla_\theta \operatorname{MSE}(D, \theta_t)$$

The gradient can be written in matrix form as

$$\nabla_\theta \operatorname{MSE}(D, \theta_t) = \tilde{\Phi}_t^\top (f_t - y)$$

where

$$\tilde{\Phi}_t = \left(-\nabla_\theta f(x_i;\theta_t)^\top -\right)_{i\le n} \in \mathbb{R}^{n\times|\theta|} \quad \text{and} \quad f_t = (f(x_i;\theta_t))_{i\le n} \in \mathbb{R}^n$$

Here $\tilde{\Phi}_t$ is the feature matrix whose i-th row is the gradient of the network output with respect to all parameters, evaluated at data point x_i and current parameters θ_t.

In the *lazy training regime* (small learning rate, wide network), the parameters stay close to initialization, so we can linearize the network via a first-order Taylor expansion around θ_0:

$$f_t \approx f_0 + \tilde{\Phi}_0 (\theta_t - \theta_0)$$

Assuming the feature matrix $\tilde{\Phi}_t$ remains approximately constant at $\tilde{\Phi}_0$ (which holds when $m \to \infty$), gradient flow yields

$$\theta_t - \theta_0 = \tilde{\Phi}_0^\top \left(\tilde{\Phi}_0\tilde{\Phi}_0^\top\right)^{-1} (f_t - f_0)$$

This says the parameter change lies in the span of the gradients and is chosen to optimally fit the training residuals.

Now let x_* be a test point. Plugging the linearization into the prediction yields

$$f_t\left(x_*\right) \approx f_0\left(x_*\right)+\nabla_\theta f\left(x_*, \theta_0\right)^{\top} \tilde{\Phi}_0^{\top}\left(\tilde{\Phi}_0 \tilde{\Phi}_0^{\top}\right)^{-1}\left(f_t-f_0\right)$$

Define the *neural tangent kernel (NTK) K* with entries

$$K_{ij}=\nabla_\theta f(x_i, \theta_0)^{\top} \nabla_\theta f(x_j, \theta_0)=\left[\tilde{\Phi}_0 \tilde{\Phi}_0^{\top}\right]_{ij}$$

and similarly $k(x_*)=\left(\nabla_\theta f(x_*, \theta_0)^{\top} \nabla_\theta f(x_i, \theta_0)\right)_{i \leq n}$.

In the infinite-width limit ($m \to \infty$), the random initialization ensures $f_0(x) \to 0$ for all x (the outputs average out), and after infinite training time ($t \to \infty$), gradient descent drives the training residual to zero so $f_t \to y$. The prediction becomes

$$f_\infty\left(x_*\right)=k\left(x_*\right)^{\top} K^{-1} y$$

This is exactly the result we obtained in Act 3 for kernel regression.

Conclusion: Gradient descent on a very wide neural network operates in a *kernel regime*, where training is equivalent to kernel ridge regression with the neural tangent kernel. The NTK is determined by the architecture and activation function, but the solution has the same closed-form structure $k(x_*)^{\top} K^{-1} y$ as any other kernel method. In practice, finite-width networks can escape this regime and learn richer, feature-learning representations—this lazy limit is a useful theoretical baseline.

4.5 Some Things from the Slides

MAP, ERM, and the conditional mean. For squared loss and any hypothesis class rich enough to contain the regression function, the Bayes-optimal solution is the conditional expectation $f^\star(x)=\mathbb{E}[Y \mid X=x]$, i.e.

$$f^\star \in \arg\min_f \mathbb{E}_{X,Y}\left[(Y-f(X))^2\right].$$

In practice $P(Y \mid X)$ is unknown, so we either (i) postulate a parametric model and perform maximum likelihood / MAP (e.g., assume $Y \mid X \sim \mathcal{N}(f_\beta(X), \sigma^2)$ and maximize $\sum_i \log p(y_i \mid x_i, \beta)$) or (ii) minimize the empirical risk $\sum_i (y_i - f(x_i))^2$ directly. For well-behaved models these two routes coincide, which explains why classical ERM with squared loss reproduces the MAP estimator of a Gaussian noise model.

Gauss–Markov optimality. Ordinary least squares does not merely give *a* solution, it gives the best linear unbiased estimator (BLUE). Consider any linear estimator $\tilde{\theta}=c^{\top} y$ that is unbiased for $a^{\top}\beta$ (i.e., $\mathbb{E}[\tilde{\theta}]=a^{\top}\beta$). The Gauss–Markov theorem states

$$\operatorname{Var}(a^{\top}\hat{\beta}) \leq \operatorname{Var}(\tilde{\theta}),$$

where $\hat{\beta}=(X^{\top}X)^{-1}X^{\top}y$ is the OLS solution and a is an arbitrary vector. Intuitively, any other unbiased linear estimator differs from OLS by an additive linear operator D with $a^{\top}DX=0$, which only inflates variance through the positive semi-definite term $DD^{\top}$. This reinforces the motivation for OLS (or its regularized cousins) when unbiasedness and linearity are desired.

Bias–variance decomposition. Suppose we highlight a specific input x and view the learned regressor $\hat{f}$ as a random variable (due to sampling various training sets). The expected squared prediction error decomposes into variance, bias, and irreducible noise:

$$\mathbb{E}_D \mathbb{E}_{Y \mid X=x}\left[(\hat{f}(x)-Y)^2\right]=\underbrace{\mathbb{E}_D\left[(\hat{f}(x)-\mathbb{E}_D[\hat{f}(x)])^2\right]}_{\text{variance}}+\underbrace{\left(\mathbb{E}_D[\hat{f}(x)]-\mathbb{E}[Y \mid X=x]\right)^2}_{\text{bias}^2}+\underbrace{\operatorname{Var}(Y \mid X=x)}_{\text{noise}}.$$

Low-capacity models (small hypothesis classes) produce high bias and low variance, while expressive models produce the opposite. Managing this trade-off is the crux of regularization and model selection.

Shrinkage beyond ridge and lasso. Ridge (ℓ_2) and lasso (ℓ_1) penalties are instances of a broader shrinkage family

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{d} x_{i,j}\beta_j \right)^2 + \lambda \sum_{j=1}^{d} |\beta_j|^q, \quad q \in (0, \infty].$$

Varying q changes the geometry of the constraint set: $q = 2$ yields spherical ridge contours, $q = 1$ produces diamond-shaped lasso constraints whose corners encourage sparsity, and $q < 1$ (non-convex) promotes even stronger sparsity at the cost of more difficult optimization. These shrinkage priors can be interpreted as MAP estimators with different prior distributions on β (Gaussian, Laplace, etc.) and help calibrate the bias–variance compromise by shrinking noisy coefficients toward zero. In practice we trace the coefficient paths as the tuning parameter (either λ or the effective degrees of freedom) varies, and select the desired amount of shrinkage via cross-validation on held-out data. Ridge paths shrink smoothly without hitting zero, whereas lasso paths *do* cross zero, enabling feature selection alongside regularization.

5 Representations

Machine learning algorithms only see the world through the representations we choose. Good representations clarify the objective (expected risk), guide the empirical procedures we rely on (empirical risk and test evaluation), and encode the structure of the data via adequate feature, scale, and mathematical spaces.

5.1 Expected vs. empirical risk

Given a hypothesis $f \in \mathcal{C}$, a loss function Q and random variables (X, Y), the conditional expected risk is

$$R(f, X) = \int Q(Y, f(X))\,\mathbb{P}(Y \mid X)\,dY,$$

while the total expected risk integrates over the data distribution

$$R(f) = \int R(f, X)\,\mathbb{P}(X)\,dX = \iint Q(Y, f(X))\,\mathbb{P}(X, Y)\,dX\,dY.$$

Learning is phrased as minimizing $R(f)$, but we only have data. With a training sample $Z_{\text{train}} = \{(X_i, Y_i)\}_{i=1}^{n}$, the empirical risk (training error) of an estimator $\hat{f}_n$ is

$$\widehat{R}(\hat{f}_n, Z_{\text{train}}) = \frac{1}{n}\sum_{i=1}^{n} Q(Y_i, \hat{f}_n(X_i)), \qquad \hat{f}_n \in \arg\min_{f \in \mathcal{C}} \widehat{R}(f, Z_{\text{train}}).$$

The test set $Z_{\text{test}} = \{(X_j, Y_j)\}_{j=n+1}^{n+m}$ yields an unbiased estimate of $R(\hat{f}_n)$ *only* if it is held out until the final estimator is fixed:

$$\widehat{R}(\hat{f}_n, Z_{\text{test}}) = \frac{1}{m}\sum_{j=n+1}^{n+m} Q(Y_j, \hat{f}_n(X_j)).$$

Whenever we adapt the estimator or its hyperparameters using the test set, we introduce dependencies between the model and the data and obtain a too optimistic risk estimate. Statistical learning therefore mandates the strict separation of training, validation, and testing.

5.2 Comparing algorithms on shared test data

To compare two learning algorithms $A^{(I)}$ and $A^{(II)}$ on the same dataset, we evaluate their paired losses on each test point j:

$$\Delta_j = Q(Y_j, \hat{f}_n^{(I)}(X_j)) - Q(Y_j, \hat{f}_n^{(II)}(X_j)), \qquad j = n+1, \ldots, n+m.$$

The sample mean $\overline{\Delta}$ and standard deviation $\operatorname{std}(\Delta)$ of these paired differences quantify which model is better. If $\overline{\Delta} - 2\,\operatorname{std}(\Delta) > 0$, then algorithm $A^{(II)}$ is reliably superior to $A^{(I)}$. This "first compare, then average" rule mirrors paired t-tests and guards against spurious conclusions from aggregate metrics alone.

5.3 Data, feature, and measurement spaces

Representation begins with deciding *what* we measure:

- **Object space $\mathcal{O}$.** Objects (digits, patients, sounds) form the domain of discourse.
- **Measurements X.** A measurement maps tuples of objects into a codomain K: $X : \mathcal{O}^{(1)} \times \cdots \times \mathcal{O}^{(R)} \to K$. Measurements can be direct (pixel intensities) or derived (edges, mel-cepstral coefficients).
- **Feature space $\mathcal{X}$.** Selecting $\mathcal{X}$ fixes the admissible metric and invariances. Numeric $\mathbb{R}^d$, Boolean, or categorical spaces each encode different similarity notions.

Typical data types emerge from the arity of the measurement:

Monadic/vectorial: $X : \mathcal{O} \to \mathbb{R}^d$ for temperature maps or feature vectors.

Dyadic: $X : \mathcal{O}^{(1)} \times \mathcal{O}^{(2)} \to \mathbb{R}$ for user–item interactions or contingency tables.

Pairwise similarity: $X : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$ for protein alignment scores or image patch similarities.

Polyadic/multiway: $X : \mathcal{O}^{(1)} \times \mathcal{O}^{(2)} \times \mathcal{O}^{(3)} \to \mathbb{R}$ such as person $\times$ behavior $\times$ trait tensors or preferential choice data.

The taxonomy clarifies whether the learner should expect absolute values, co-occurrence counts, or structured relational inputs.

5.4 Scale types and transformation invariances

Scale choices express which transformations should leave conclusions invariant:

- **Nominal scale:** only equality matters; any bijection preserves meaning.
- **Ordinal scale:** rankings are meaningful; order-preserving functions are admissible.
- **Interval scale:** information resides in differences; affine transformations $f(x) = ax + c$ with $a > 0$ are allowed (e.g., Fahrenheit).
- **Ratio scale:** differences and ratios matter; only scalings $f(x) = ax$, $a > 0$ preserve structure (e.g., Kelvin).
- **Absolute scale:** literal values matter; only the identity transformation is valid (e.g., exam grades).

Data whitening—scaling features by their standard deviation—is one practical way to enforce comparable dynamic ranges so that the chosen metric respects the intended invariances.

5.5 Mathematical spaces underlying representations

Different mathematical spaces formalize increasingly rich structures:

- **Topological spaces** $(X, \mathcal{J})$**:** $\mathcal{J}$ is a family of subsets containing X and $\varnothing$, closed under finite intersections and arbitrary unions. Topology captures neighborhood relations without prescribing distances.
- **Metric spaces** (X, d)**:** A metric d satisfies non-negativity, identity of indiscernibles, symmetry, and the triangle inequality (or the stronger ultrametric inequality). Metrics quantify distances and therefore induce topologies.
- **Euclidean vector spaces** (V, ϕ)**:** A vector space equipped with a scalar product ϕ satisfying distributivity, symmetry, homogeneity, and positive definiteness. The induced norm $\|x\| = \sqrt{\phi(x, x)}$ generalizes standard Euclidean geometry.

Every Euclidean space is metric and thus topological, but the converse need not hold. Representation design should match the amount of reliable structure (topological, metric, or vectorial) that the measurements truly support.

5.6 Probability spaces

Probability theory provides the formal language for risk:

- **Sample space** $\Omega = \{\omega_1, \ldots, \omega_N\}$ collects all elementary outcomes (e.g., sequences of coin flips).
- **Event algebra** $\mathcal{A} \subseteq 2^\Omega$ contains Ω and is closed under union, intersection, and set difference, so that compound statements about outcomes remain valid events.

- **Probability measure** $\mathbb{P} : \mathcal{A} \to [0, 1]$ assigns weights $p(\omega_i)$ to elementary events, satisfies $\mathbb{P}(\Omega) = 1$, and extends additively to events $A \in \mathcal{A}$ via $\mathbb{P}(A) = \sum_{\omega_i \in A} p(\omega_i)$.

The triple $(\Omega, \mathcal{A}, P)$ underpins expected risk: once we define the events (representations) and their probabilities, we can integrate losses and reason about generalization.

6 Gaussian Processes for Regression

Gaussian processes (GPs) turn Bayesian linear regression into a flexible, non-parametric model that reasons about functions via distributions over all possible outputs. The slides emphasize three pillars: (i) the Bayesian linear regression foundation, (ii) kernel engineering for encoding inductive biases, and (iii) prediction, validation, and application pipelines that exploit GP uncertainty.

6.1 From Bayesian linear regression to GPs

Linear regression with Gaussian noise assumes $Y = x^\top\beta + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Placing a Gaussian prior on the weights, $\beta \sim \mathcal{N}(0, \Lambda^{-1})$, yields the posterior

$$p(\beta \mid X, y) = \mathcal{N}\big(\mu_\beta, \Sigma_\beta\big), \qquad \mu_\beta = \big(X^\top X + \sigma^2\Lambda\big)^{-1} X^\top y, \quad \Sigma_\beta = \sigma^2\big(X^\top X + \sigma^2\Lambda\big)^{-1}.$$

Understanding the posterior: The posterior mean μ_β combines the data (via $X^\top X$ and $X^\top y$) with the prior precision Λ. This is exactly the ridge regression solution we saw in Chapter 4.2, where Λ acts as a regularization matrix (compare to equation 2). The term $X^\top X + \sigma^2\Lambda$ plays the role of a regularized Hessian, stabilizing the inversion. The posterior covariance Σ_β (compare to equation 3) captures uncertainty about β: directions along which the data is uninformative (small eigenvalues of $X^\top X$) retain their prior uncertainty, while directions well-explained by data shrink toward zero. Note that all uncertainty decreases with sample size n (more rows in X), and the marginal posterior variance of the i-th weight is Σ_β^{ii}.

Observations are linear combinations of Gaussian variables, so the vector of outputs $y = X\beta + \varepsilon$ is jointly Gaussian with mean zero and covariance

$$\mathrm{Cov}(y) = X\Lambda^{-1}X^\top + \sigma^2 I_n.$$

Derivation. Since $\beta \sim \mathcal{N}(0, \Lambda^{-1})$ and $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_n)$ are independent,

$$\begin{aligned}
\mathbb{E}[y] &= \mathbb{E}[X\beta + \varepsilon] = X\,\mathbb{E}[\beta] + \mathbb{E}[\varepsilon] = 0,\\
\mathrm{Cov}(y) &= \mathbb{E}\big[(y - \mathbb{E}[y])(y - \mathbb{E}[y])^\top\big]\\
&= \mathbb{E}\big[(X\beta + \varepsilon)(X\beta + \varepsilon)^\top\big]\\
&= X\,\mathbb{E}[\beta\beta^\top]X^\top + \mathbb{E}[\varepsilon\varepsilon^\top] + X\,\mathbb{E}[\beta\varepsilon^\top] + \mathbb{E}[\varepsilon\beta^\top]X^\top\\
&= X\,\Lambda^{-1}X^\top + \sigma^2 I_n,
\end{aligned}$$

where we used $\mathbb{E}[\beta\beta^\top] = \mathrm{Cov}(\beta) = \Lambda^{-1}$ for a zero-mean Gaussian. The cross terms vanish because $\mathbb{E}[\beta\varepsilon^\top] = \mathbb{E}[\beta]\mathbb{E}[\varepsilon]^\top = 0$ and $\mathbb{E}[\varepsilon\beta^\top] = \mathbf{0}$ by independence and zero means.

Defining $k(x_i, x_j) = x_i^\top\Lambda^{-1}x_j$ turns this covariance into a kernel (Gram) matrix K, so $y \sim \mathcal{N}(0, K + \sigma^2 I)$. Replacing the dot-product kernel by any valid positive semi-definite kernel function "kernelizes" Bayesian ridge regression; the resulting stochastic process over functions is a Gaussian process.

6.2 What is a Gaussian process?

A **Gaussian process (GP)** is a collection of random variables indexed by inputs (e.g., $x \in \mathbb{R}^d$) such that any finite subset has a joint Gaussian distribution. Equivalently, a GP is a *distribution over functions.* We write

$$f \sim \mathcal{GP}\big(m, k\big), \quad m(x) = \mathbb{E}[f(x)], \quad k(x, x') = \mathrm{Cov}(f(x), f(x')).$$

For any inputs $X = [x_1, \ldots, x_n]$, the function values satisfy

$$f(X) = [f(x_1), \ldots, f(x_n)]^\top \sim \mathcal{N}\big(m(X),\ K\big), \quad K_{ij} = k(x_i, x_j).$$

With Gaussian observation noise $y = f(X) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$, we obtain

$$y \sim \mathcal{N}\big(m(X),\ K + \sigma^2 I\big).$$

The mean function m and kernel k fully specify the prior over functions: the kernel encodes smoothness, invariances, and correlation structure. Conditioning the prior GP on data (X, y) yields a *posterior GP* with updated mean and covariance, whose predictive mean/variance coincide with the closed-form formulas presented below. This makes GPs a principled, non-parametric way to model functions with calibrated uncertainty.

6.3 Kernel design

Kernels encode similarity and thereby the structure of the functions we wish to learn:

- Valid kernels must be symmetric and generate positive semi-definite Gram matrices for any finite set of inputs. They implicitly act as inner products in (possibly infinite-dimensional) Hilbert spaces.
- Standard kernels on $\mathbb{R}^d$ include linear $k(x, x') = x^\top x'$, polynomial $k(x, x') = (x^\top x' + 1)^p$, squared exponential $k(x, x') = \sigma_f^2 \exp(-\|x - x'\|^2/(2\ell^2))$, rational quadratic, exponential, and periodic kernels. Each carries different invariances (smoothness, periodicity, etc.).
- Kernels compose: sums, products, positive scalings, or applying positive-coefficient polynomials / exponentials to a base kernel all preserve validity. This "kernel engineering" enables similarity measures on non-vector data such as strings, graphs (diffusion kernels), or probability distributions.

Designing an appropriate kernel is tantamount to choosing the hypothesis space for the GP.

6.4 Prediction with Gaussian processes

Given training data (X, y) and a test input $x_\star$, we are interested in $y_\star$. The joint distribution of y and $y_\star$ is Gaussian:

$$\begin{bmatrix} y \\ y_\star \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K + \sigma^2 I & k \\ k^\top & c \end{bmatrix}\right),$$

where $K = k(X, X)$, $k = k(x_\star, X)$ is the vector of cross-covariances, and $c = k(x_\star, x_\star) + \sigma^2$. This is a natural extension of the setup without $x_\star$. Conditioning a multivariate Gaussian gives the predictive distribution

$$p(y_\star \mid x_\star, X, y) = \mathcal{N}\big(\mu_\star, \sigma_\star^2\big), \quad \mu_\star = k^\top (K + \sigma^2 I)^{-1} y, \quad \sigma_\star^2 = c - k^\top (K + \sigma^2 I)^{-1} k.$$

Derivation. **Theorem (Conditional Gaussian).** Suppose

$$\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right),$$

with $\mathbf{a}_1, \mathbf{u}_1 \in \mathbb{R}^e$, $\mathbf{a}_2, \mathbf{u}_2 \in \mathbb{R}^f$, and covariance blocks $\Sigma_{11} \in \mathbb{R}^{e \times e}$, $\Sigma_{12} \in \mathbb{R}^{e \times f}$, $\Sigma_{21} \in \mathbb{R}^{f \times e}$, $\Sigma_{22} \in \mathbb{R}^{f \times f}$ positive semidefinite. Then the conditional distribution of $\mathbf{a}_2$ given $\mathbf{a}_1 = \mathbf{z}$ is

$$p(\mathbf{a}_2 \mid \mathbf{a}_1 = \mathbf{z}) = \mathcal{N}\big(\, \mathbf{u}_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{z} - \mathbf{u}_1),\ \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} \,\big).$$

Hence GPs output **both a mean prediction and an uncertainty quantification**. This is however not the real uncertainty about the function value at $x_\star$, but the uncertainty we get inside our model, which is constrained to gaussian processes with a specific kernel.

```
Algorithm Prediction with Gaussian Processes
Require: $n$ observed data ($\mathbf{X} = (x_1, \dots, x_n)^\mathsf{T} \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^n$), kernel function $k$,
  noise variance $\sigma^2$, new data point $x_{n+1} \in \mathbb{R}^d$

  $\mathbf{K} \leftarrow (k(x_i, x_j))_{1 \le i,j \le n}$                              // Compute kernel matrix
  $\mathbf{k} \leftarrow (k(x_{n+1}, x_i))_{1 \le i \le n}$                              // Similarity of new data point and observed data
  $\mu_{y_{n+1}} \leftarrow \mathbf{k}^\mathsf{T}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$                  // Mean of predictive distribution
  $\sigma^2_{y_{n+1}} \leftarrow k(x_{n+1}, x_{n+1}) - \mathbf{k}^\mathsf{T}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}$    // Variance of predictive distribution

  return $\mathcal{N}(y_{n+1} | \mu_{y_{n+1}}, \sigma^2_{y_{n+1}})$                      // Return predictive distribution
```

The prediction algorithm returns a distribution function. The prediction at x_{n+1} yields a mean value $\mu_{y_{n+1}}$ and a variance $\sigma^2_{y_{n+1}}$. Furthermore, samples y_{n+1} can be drawn from this distribution.

6.5 Validating kernels and hyperparameters

Practical GP performance depends on hyperparameters such as the length-scale ℓ, signal variance σ_f^2, or kernel choice. The lecture highlights:

- Evidence maximization (type-II ML) to optimize hyperparameters by maximizing $\log p(y \mid X, \theta)$.
- Cross-validation schemes (random splits, leave-one-out) that rank kernels by predictive performance on held-out data. Synthetic experiments show that proper scoring rules recover the data-generating kernel, while real data (e.g., power plant energy output) can exhibit different optima depending on the scoring metric (squared exponential vs. periodic kernels).
- Bayesian comparison of "teacher" and "student" kernels: evaluate how well a candidate kernel matches data generated from another kernel by comparing their posterior predictive distributions.

Kernel validation is therefore an empirical model-selection layer that complements theoretical kernel properties.

6.6 Applications: control and fMRI

Because GPs model functions with calibrated uncertainty, they are appealing for safety-critical and data-efficient applications:

- **Safe Bayesian optimization for control.** In automatic controller tuning (e.g., quadrotor flight), the unknown performance function over controller parameters is modeled as a GP. Safe optimization explores only parameter settings whose predicted performance exceeds a safety threshold with high probability, gradually expanding the safe set and converging to the global optimum using few evaluations.
- **Robust controller design.** GPs capture both the mean and variance of system responses, enabling controllers that respect safety constraints while improving performance across uncertain dynamics. Comparisons to hand-tuned controllers highlight faster convergence and higher reliability.

These case studies reinforce that GP regression is more than a curve fitting tool—it is a probabilistic modeling framework that unifies inference, kernel engineering, and safety-aware decision making.

7 Ensemble Methods

Bagging, boosting, and stacking are the three canonical ensemble strategies:

- **Bagging** (bootstrap aggregating) trains many models independently on bootstrapped samples and averages or votes their predictions to reduce variance.
- **Boosting** trains models sequentially, each focusing on the errors of the previous ones, to reduce bias (e.g., AdaBoost, gradient boosting).
- **Stacking** trains diverse base models and then fits a meta-model on their outputs to learn the best combination.

Ensemble methods combine multiple base learners to obtain a predictor whose bias, variance, or loss surface is superior to any constituent model. By averaging or sequentially correcting learners trained on diverse views of the data, ensembles produce more stable predictions, calibrated uncertainties, and richer inductive biases than single models. This chapter summarizes the bootstrap-based family (bagging and random forests), boosting, and more general stacking strategies, with an emphasis on how they target different points along the bias–variance trade-off.

7.1 Why ensembles help

Let $\hat{f}(x)$ be a single hypothesis trained on data D. Ensembles form $\hat{f}_{\text{ens}}(x) = \sum_{b=1}^{B} w_b \hat{f}_b(x)$ from base learners $\hat{f}_b$. Two canonical effects explain their success:

- **Variance reduction.** If the $\hat{f}_b$'s are identically distributed with variance σ^2 and pairwise correlation ρ, then
$$\operatorname{Var}\left[\hat{f}_{\text{avg}}(x)\right] = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2,$$
which shrinks as B increases whenever $\rho < 1$. Bagging manipulates the data (bootstrap samples) to drive correlations down and thus stabilize high-variance learners such as decision trees or neural networks trained on small data.

Derivation. For the average predictor $\hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x)$,

$$\begin{aligned}
\operatorname{Var}[\hat{f}(x)] &= \mathbb{E}_D\left(\hat{f}(X) - \mathbb{E}_D \hat{f}(X)\right)^2 \\
&= \mathbb{E}_D\left(\frac{1}{B}\sum_{i=1}^{B} \hat{f}_i(x) - \frac{1}{B}\sum_{i=1}^{B} \mathbb{E}_D \hat{f}_i(x)\right)^2 \\
&= \mathbb{E}_D\left(\frac{1}{B}\sum_{i=1}^{B}\left(\hat{f}_i(x) - \mathbb{E}_D \hat{f}_i(x)\right)\right)^2 \\
&= \frac{1}{B^2}\sum_{i=1}^{B} \operatorname{Var}_D\left[\hat{f}_i(x)\right] + \frac{1}{B^2}\sum_{i\neq j} \operatorname{Cov}\left[\hat{f}_i(x), \hat{f}_j(x)\right]
\end{aligned}$$

If all base learners share variance σ^2 and pairwise covariance $\operatorname{Cov}(\hat{f}_i, \hat{f}_j) = \rho\,\sigma^2$,

$$\begin{aligned}
\operatorname{Var}\left[\hat{f}_{\text{avg}}(x)\right] &= \frac{B}{B^2}\sigma^2 + \frac{B(B-1)}{B^2}\rho\,\sigma^2 \\
&= \frac{1}{B}\sigma^2 + \left(1 - \frac{1}{B}\right)\rho\,\sigma^2 \\
&= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.
\end{aligned}$$

Thus variance decreases like $1/B$ when correlations are small ($\rho \approx 0$), while residual correlation limits the gain.

- **Bias correction.** Sequential ensembles such as boosting fit a series of weak learners to the residuals (negative gradients) of the current model. Each stage nudges the predictor toward lower bias and can transform a barely better-than-random base learner into a strong classifier.

These mechanisms are complementary: bagging primarily attacks variance, boosting primarily attacks bias, and stacking blends heterogeneous models to capture complementary inductive biases.

7.2 Bagging and random forests

Bagging (bootstrap aggregating) trains each base learner on a bootstrap sample of the training set. For classification, the ensemble prediction is a majority vote; for regression, it is an average:

$$\hat{c}_B(x) = \operatorname{sgn}\left(\sum_{b=1}^{B} c_b(x)\right), \qquad \hat{f}_B(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x).$$

Bootstrap samples overlap but are not identical, so high-variance learners (e.g., trees) move in different directions and averaging stabilizes them.

```
**Bagging classifier**
Input: data $\{(x_i, y_i)\}_{i=1}^n$, number of models $B$
for $b = 1$ to $B$ do
    draw bootstrap sample $Z_b$ from the data (sampling with replacement)
    fit classifier $c_b$ on $Z_b$
end for
output $\hat{c}_B(x) = \operatorname{sgn}(\sum_{b=1}^{B} c_b(x))$
```

Random forests add a second source of randomness by selecting a random subset of features at each split. This further reduces correlation between trees without increasing bias too much. Each tree is a standard decision tree trained on its own bootstrap sample; at prediction time the forest aggregates all trees. Typical split criteria include Gini impurity, entropy, and misclassification rate.

Weak learners used in ensembles. In practice, weak learners can be decision stumps, full decision trees, perceptrons/MLPs, or radial basis function networks. Bagging benefits most from unstable learners whose predictions change noticeably under small perturbations of the data.

7.3 Boosting

For now we're only interested in classification.

Given a Dataset $D = \{(x_i, y_i)\}_{i=1} \subseteq \mathbb{R}^d \times \{-1, +1\} \sim p_*$, where y_i are labels and x_i are features, with an unknown distribution p_*, we want to learn a classifier $G : \mathbb{R}^d \to \{-1, +1\}$ that minimizes the expected 0–1 loss:

$$L(G) = \mathbb{E}_{(X,Y)\sim p_*}[\mathbb{I}\{G(X) \neq Y\}]$$

there are three problems with this formulation:

- The distribution p_* is unknown, so we cannot compute the expected loss directly.
- The 0–1 loss is not differentiable, making optimization difficult.
- The hypothesis space of all classifiers $G : \mathbb{R}^d \to \{-1, +1\}$ is too large to search over.

To address these issues, we make the following changes:

- We replace the expected loss with the empirical loss on the training data:

$$\hat{L}(G, D) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{G(x_i) \neq y_i\}$$

- We replace the 0–1 loss with a surrogate loss function $\ell : \mathbb{R} \to \mathbb{R}_+$ that is differentiable and convex, such as the exponential loss $\ell(z) = e^{-yG(x)}$.

- We restrict our hypothesis space to a set of weak learners $\mathcal{H}^A$ (A for additive), with

$$\mathcal{H}^A = \left\{ G_m \mid G_m(x) = \sum_{i \leq m} \beta_i f_i(x), f_i \in \mathcal{H}, \beta_i \in \mathbb{R} \right\}$$

This won't work with linear models. But with weak learners (e.g., decision stumps), we can use boosting to iteratively build a strong classifier by adding weak learners that minimize the surrogate loss on the training data. The additive form also makes it natural to update the model stage by stage.

So then our objective becomes:

$$\min_{G \in \mathcal{H}^A} \frac{1}{n} \sum_{i \leq n} \mathcal{L}(y_i; G(x_i))$$

with

$$\mathcal{L}(y_i; G(x)) = \exp(-y_i G(x_i)) = \exp(-y_i G_{m-1}(x_i)) \exp(-y_i \beta_m f_m(x_i)) = w_i^{m-1} \mathcal{L}(y_i; \beta_m f_m(x_i))$$

because $G_m(x) = G_{m-1}(x) + \beta_m f_m(x)$. What did we do? We decomposed the loss at step m into the loss at step $m-1$ and the contribution of the new weak learner f_m with weight β_m. So now we have two things to optimize: the weak learner f_m and its weight β_m.

and we can rewrite the objective as:

$$\min_{G_{m-1} \in \mathcal{H}^A_{m-1}} \min_{f_m \in \mathcal{H}, \beta_m \in \mathbb{R}} \sum_{i \leq n} w_i^{m-1} \mathcal{L}(y_i; \beta_m f_m(x_i))$$

7.4 Forward Stagewise Additive Modeling (FSAM)

FSAM provides the bridge from the additive objective to an actual algorithm. Instead of solving for all f_m and β_m at once, it builds the model stage by stage: each step adds the single component that most improves the current objective. This is both computationally simple and conceptually aligned with boosting's idea of sequential correction. At each step m, we fix the previous model G_{m-1} and optimize for the new weak learner f_m and its weight β_m:

```
$G_0(x) \leftarrow 0$, $w_i^0 \leftarrow 1/n$ for all $i$
for $m = 1$ to $M$ do
    $\min_{f_m \in \mathcal{H}, \beta_m \in \mathbb{R}} \sum_{i \leq n} w_i^{m-1} \mathcal{L}(y_i; \beta_m f_m(x_i))$
    $G_m(x) \leftarrow G_{m-1}(x) + \beta_m f_m(x)$
    $w_i^m \propto w_i^{m-1} \exp(-y_i \beta_m f_m(x_i))$
end for
```

Future classifiers try to correct the mistakes of past classifiers by focusing more on misclassified examples (increasing their weights). This way, the ensemble learns from its errors and improves over time. Gradient boosting generalizes this stagewise procedure by choosing updates from the negative gradient of an arbitrary differentiable loss.

7.5 Gradient Descent Boosting

FSAM still depends on a particular loss and reweighting scheme. Gradient boosting generalizes the same stagewise idea to any differentiable loss by viewing boosting as gradient descent in function space. The key move is to replace the weighted classification objective with a gradient step on the loss with respect to the current model's predictions. Wanted $G_\theta(x)$ where $\theta \in \Theta$ are parameters. We can use gradient descent to minimize the empirical loss:

```
$\theta_0$ given
for $t = 1$ to $T$ do
    $\theta_t \leftarrow \theta_{t-1} - \alpha_t \nabla_\theta \mathcal{L}(D; \theta_{t-1})$
end for
```

At the end

$$\theta_T = \theta_0 - \sum_{t \leq T} \alpha_t \nabla_\theta \mathcal{L}(D; \theta_{t-1}).$$

The combined algorithm looks like this when we perform gradient descent in function space and fit weak learners to the negative gradients (pseudo-residuals). This reveals why boosting works for a wide range of losses and why the base learner only needs to approximate the negative gradient at each step:

```
$G_0(x) \leftarrow \arg\min_\gamma \sum_i \mathcal{L}(y_i;\gamma)$
**for** $t = 1$ to $m$ **do**
    $r_i^{t-1} \leftarrow -\left[\frac{\partial \mathcal{L}(y_i;G(x_i))}{\partial G(x_i)}\right]_{G(x)=G_{t-1}(x)}$ for all $i$
    $\min_{f_t\in\mathcal{H},\beta_t\in\mathbb{R}} \sum_{i\le n}(r_i^{t-1} - \beta_t f_t(x_i))^2$
    $G_t(x) \leftarrow G_{t-1}(x) + \beta_t f_t(x)$
**end for**
```

This recovers gradient boosting: each stage fits a weak learner to the pseudo-residuals and updates the additive model. Small step sizes and shallow trees typically improve generalization. With exponential loss and binary weak learners, this reduces to AdaBoost.

7.6 AdaBoost

AdaBoost is the classic instantiation of the above ideas: it chooses the exponential loss, uses binary weak learners, and yields simple closed-form updates. The reweighting scheme is not arbitrary; it is exactly what makes each new classifier focus on mistakes while keeping the additive model interpretable. AdaBoost is a concrete instance of gradient boosting (and FSAM) with exponential loss and binary labels. Let $y_i \in \{-1,+1\}$ and $c_m(x) \in \{-1,+1\}$. At step m, the weighted error is

$$\varepsilon_m = \frac{\sum_{i=1}^n w_i\,\mathbb{I}\{c_m(x_i) \neq y_i\}}{\sum_{i=1}^n w_i}.$$

The update weight is

$$\alpha_m = \tfrac{1}{2}\log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right),$$

and the data weights are updated by

$$w_i \leftarrow w_i \exp\bigl(-\alpha_m y_i c_m(x_i)\bigr), \qquad \sum_i w_i = 1.$$

The final classifier is the sign of the weighted vote, $\hat{c}(x) = \operatorname{sgn}(\sum_m \alpha_m c_m(x))$.

Derivation. The choice of α_m follows from minimizing the weighted exponential loss for a fixed classifier c_m:

$$\sum_i w_i \exp\bigl(-\alpha y_i c_m(x_i)\bigr) = (1-\varepsilon_m)e^{-\alpha} + \varepsilon_m e^{\alpha}.$$

Differentiating and setting to zero yields $-(1-\varepsilon_m)e^{-\alpha} + \varepsilon_m e^{\alpha} = 0$, hence

$$\alpha_m = \tfrac{1}{2}\log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right).$$

The algorithm only requires each weak learner to perform slightly better than chance ($\varepsilon_m < 1/2$). If a learner is worse than chance, flipping its predictions reduces the error.

7.7 Boosting as additive modeling

Beyond the algorithm, it is useful to understand what boosting is optimizing in the population. This perspective ties the stagewise updates to statistical decision theory and explains why boosting often improves margins. AdaBoost can be viewed as fitting an additive model by minimizing the exponential loss

$$J(F) = \mathbb{E}\bigl[\exp(-yF(x))\bigr].$$

The minimizer of this loss is proportional to the log-odds of the class probabilities, so $F(x)$ estimates a scaled log posterior ratio.

Derivation. Condition on x and write $p = \mathbb{P}(y = 1 \mid x)$. Then

$$\mathbb{E}\Big[e^{-yF(x)} \mid x\Big] = p\,e^{-F(x)} + (1-p)\,e^{F(x)}.$$

Differentiating with respect to $F(x)$ and setting to zero gives

$$-p\,e^{-F(x)} + (1-p)\,e^{F(x)} = 0 \quad \Rightarrow \quad F(x) = \tfrac{1}{2}\log\left(\frac{p}{1-p}\right).$$

Thus the optimal $F(x)$ is a log-odds function, linking AdaBoost to additive logistic regression.

This perspective explains why boosting often improves margins and why it can be interpreted as stagewise functional optimization.

7.8 Stacking

Stacking combines heterogeneous models by training a meta-learner on their predictions. To avoid information leakage, base-model predictions for the meta-learner are typically generated via cross-validation. This lets the meta-learner discover which models are reliable in different regions of the input space, often outperforming any single model or uniform averaging.

8 Convex Optimization

Convex optimization studies problems whose objective and feasible sets are convex, guaranteeing that every local optimum is also global. These structure-driven guarantees let us design algorithms with provable convergence, quantify sensitivity, and provide certificates of optimality via dual variables. This chapter outlines the key definitions, canonical problem classes, duality theory, and foundational algorithms emphasized in the lecture notes.

8.1 Convex sets, functions, and problems

Let $C \subseteq \mathbb{R}^d$ be a set. C is **convex** if for any $x, y \in C$ and $\theta \in [0,1]$, the convex combination $\theta x + (1-\theta)y$ lies in C. Typical convex sets are affine subspaces, halfspaces, Euclidean balls, ellipsoids, spectrahedra, and probability simplices.

A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if its domain is convex and for every x, y and $\theta \in [0,1]$

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y).$$

When f is differentiable, convexity is equivalent to the first-order inequality

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x),$$

meaning the affine tangent is a global under-estimator. If f is twice differentiable, convexity is equivalent to $\nabla^2 f(x) \succeq 0$ for all x in the interior of the domain. **Strong convexity** with parameter $m > 0$ strengthens the inequality to

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{m}{2}\|y - x\|^2,$$

implying a unique minimizer and improved convergence rates.

A constrained optimization problem minimizes an objective subject to equality and inequality constraints. It is convex when the objective and inequality constraints are convex and the equality constraints are affine, so the feasible set is convex.

A generic convex optimization problem reads

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, \quad i = 1, \dots, m, \\ & Ax = b, \end{aligned}$$

where each f_i is convex and the equality constraints define an affine set. The feasible set $\{x \mid f_i(x) \leq 0, Ax = b\}$ must be nonempty for the problem to be well-posed. Convexity ensures that any Karush–Kuhn–Tucker (KKT) point is globally optimal.

8.2 Illustrative example: closest point on a disk

Consider a drone at (x_0, y_0, z_0) and a person restricted to the flat disk $\{(x, y, 0) \mid x^2 + y^2 \leq r\}$. The closest point solves

$$\min_{x,y,z} \ \tfrac{1}{2}\|(x,y,z) - (x_0,y_0,z_0)\|_2^2 \quad \text{s.t.} \quad z = 0,\ x^2 + y^2 \leq r.$$

Let $f(x,y,z) = \frac{1}{2}\|(x,y,z) - (x_0,y_0,z_0)\|_2^2$, $g(x,y,z) = z$, and $h(x,y,z) = x^2 + y^2 - r$. Then

$$\begin{aligned} \nabla f(x,y,z) &= (x - x_0, y - y_0, z - z_0), \\ \nabla g(x,y,z) &= (0,0,1), \\ \nabla h(x,y,z) &= (2x, 2y, 0). \end{aligned}$$

At an optimum, there exist multipliers $\lambda \in \mathbb{R}$ and $\alpha \geq 0$ such that

$$\nabla f(x,y,z) + \lambda \nabla g(x,y,z) + \alpha \nabla h(x,y,z) = 0,$$

together with $z = 0$, $x^2 + y^2 \leq r$, and $\alpha(x^2 + y^2 - r) = 0$. This captures whether the closest point lies in the interior of the disk ($\alpha = 0$) or on the boundary ($\alpha > 0$).

8.3 Standard convex programs in machine learning

Many estimators from earlier chapters fit this template:

- **Least squares / ridge regression:** $f(x) = \frac{1}{2}\|Ax - b\|_2^2 + \frac{\lambda}{2}\|x\|_2^2$ is convex quadratic; ridge adds strong convexity, yielding the closed form in equation (2).
- **Lasso:** minimize $\frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$. The ℓ_1 norm promotes sparsity via a polyhedral penalty while keeping the problem convex. Soft-thresholding is the proximal operator driving coordinate descent.
- **Support Vector Machines (SVM):** hinge-loss objectives $\sum_i \max(0, 1 - y_i w^\top x_i)$ with ℓ_2 regularization define convex problems whose dual has sparse support vectors.
- **Logistic regression:** the negative log-likelihood $\sum_i \log(1+\exp(-y_i w^\top x_i))$ is convex. Adding ℓ_1 or ℓ_2 penalties yields generalized linear models solvable via gradient or Newton methods.
- **Matrix completion:** minimizing $\sum_{(i,j)\in\Omega}(X_{ij} - M_{ij})^2 + \lambda\|X\|_*$ uses the nuclear norm (convex surrogate of rank) to recover low-rank matrices.

Specialized subfamilies often admit faster algorithms:

- **Linear programs (LP):** $f_0(x) = c^\top x$, $f_i(x) = a_i^\top x - b_i$.
- **Quadratic programs (QP):** quadratic objective with linear constraints.
- **Second-order cone programs (SOCP)** and **semidefinite programs (SDP)** capture norms and PSD constraints, respectively, and power robust control plus covariance fitting.

8.4 Lagrangian duality and KKT conditions

For

$$\min_x f_0(x) \quad \text{s.t.} \quad f_i(x) \le 0,\ Ax = b,$$

introduce multiplier $\lambda \ge 0$ for inequalities and ν for equalities. The **Lagrangian** is

$$\mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^\top(Ax - b)$$

Intuitively, it adds weighted penalties for constraint violations to the objective: λ (with $\lambda \ge 0$) acts like a price for violating $f_i(x) \le 0$, while ν enforces the equality $Ax = b$. The saddle-point view is that we minimize over x but maximize over multipliers (λ, ν), so any violation is punished at the optimum, yielding feasibility and the KKT conditions.

The Lagrangian lower-bounds the primal objective: $g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) \le f_0(x)$ for all feasible x. The **dual problem** maximizes $g(\lambda, \nu)$ subject to $\lambda \ge 0$. Weak duality ($g \le p^\star$) always holds, where $p^\star$ is the optimal primal value, while **strong duality** ($g^\star = p^\star$) holds under mild constraint qualifications such as Slater's condition (strict feasibility: there exists x with $f_i(x) < 0$ and $Ax = b$). Dual variables often have sensitivity interpretations, e.g., the effect of tightening constraints. The dual trades the original constraints for a typically more complex objective but a simpler feasible set, which can make problems like SVMs easier to solve.

The **KKT conditions** describe optimality when strong duality holds:

$$\begin{aligned}
&\text{Primal feasibility: } f_i(x^\star) \le 0,\ Ax^\star = b.\\
&\text{Dual feasibility: } \lambda^\star \ge 0.\\
&\text{Stationarity: } \nabla f_0(x^\star) + \sum_i \lambda_i^\star \nabla f_i(x^\star) + A^\top \nu^\star = 0.\\
&\text{Complementary slackness: } \lambda_i^\star f_i(x^\star) = 0.
\end{aligned}$$

For unconstrained problems these reduce to $\nabla f_0(x^\star) = 0$, consistent with calculus. In SVMs, for example, KKT implies that only points on the margin have non-zero dual variables, explaining sparsity in the dual solution.

8.5 Solving convex optimization problems

When Slater's condition holds, the KKT conditions are necessary and sufficient. A direct strategy is to solve the KKT system (stationarity, primal feasibility, dual feasibility, and complementary slackness) for $x^\star$, $\lambda^\star$, and $\nu^\star$. If this system is intractable, solve the dual problem instead; under strong duality, primal solutions can be recovered from the dual optimum using stationarity and complementary slackness.

9 Support Vector Machines

For support vector machines (SVMs), we're interested in maximising the margin between classes. So given a dataset $D = \{(x_i, y_i)\}$ with $y_i \in \{-1, +1\}$ and guaranteed linear separability, we want to find a hyperplane defined by (w, w_0) such that $w^T x_i + w_0 > 0$ if $y_i = 1$ and $w^T x_i + w_0 < 0$ if $y_i = -1$. Or in other terms, we want to satisfy the constraints $y_i(w^T x_i + w_0) > 0$ for all i. The margin is defined as the distance from the hyperplane to the closest data point, which can be expressed as $\frac{2}{\|w\|}$ (*see derivation below*).

Derivation. Given the hyperplane $\{x \mid w^\top x + w_0 = 0\}$, the signed distance of any point x to the hyperplane is

$$\mathrm{dist}(x, \mathcal{H}) = \frac{w^\top x + w_0}{\|w\|}.$$

To see why, let x_0 be any point on the hyperplane, so $w^\top x_0 + w_0 = 0$. The vector w is normal to the hyperplane, hence the shortest path from x to $\mathcal{H}$ is along the unit normal $w/\|w\|$. The signed distance is the projection of $x - x_0$ onto this unit normal:

$$\frac{w^\top (x - x_0)}{\|w\|} = \frac{w^\top x + w_0}{\|w\|}.$$

Because $y_i \in \{-1, +1\}$, we can enforce the scale of (w, w_0) by requiring that the closest points satisfy $y_i(w^\top x_i + w_0) = 1$. Under this normalization, there exist parallel "margin" hyperplanes

$$w^\top x + w_0 = 1 \quad \text{and} \quad w^\top x + w_0 = -1$$

touching the positive and negative classes, respectively. The perpendicular distance between these two planes is the geometric margin:

$$\gamma = \frac{1 - (-1)}{\|w\|} = \frac{2}{\|w\|}.$$

Maximizing γ therefore amounts to minimizing $\|w\|$ (or equivalently $\frac{1}{2}\|w\|^2$ for convenience) under the constraints $y_i(w^\top x_i + w_0) \geq 1$ for all i. This yields the hard-margin SVM formulation

$$\min_{w, w_0} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y_i(w^\top x_i + w_0) \geq 1, \ i = 1, \ldots, n,$$

whose solution maximizes the separating margin.

9.1 Slater's Condition

By assumption that the data is linearly separable, there exists a feasible point (w, w_0) satisfying $y_i(w^\top x_i + w_0) > 1$ for all i. This strictly feasible point ensures that Slater's condition holds, guaranteeing strong duality between the primal and dual SVM problems.

9.2 The Dual

Writing the hard-margin primal in standard form,

$$\begin{aligned} \min_{w, w_0} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w^\top x_i + w_0) - 1 \geq 0, \quad i = 1, \ldots, n, \end{aligned}$$

introduce Lagrange multipliers $\alpha_i \geq 0$ for each margin constraint. The Lagrangian is

$$\mathcal{L}(w, w_0, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \big(y_i(w^\top x_i + w_0) - 1\big).$$

To form the dual we minimize $\mathcal{L}$ over the primal variables. Setting derivatives to zero yields the stationarity conditions

$$\nabla_w \mathcal{L} = w - \sum_i \alpha_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_i \alpha_i y_i x_i,$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = -\sum_i \alpha_i y_i = 0.$$

Substituting back gives the dual objective

$$g(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^\top x_j = \sum_{i=1} \alpha_i - \frac{1}{2}||w(\alpha)||^2,$$

where $w(\alpha) = \sum_i \alpha_i y_i x_i$ is the primal weight vector written in terms of the dual variables. The constraints are $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$. Hence the dual problem is the quadratic program

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \sum_{i=1} \alpha_i - \frac{1}{2}||w(\alpha)||^2 \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, n, \\ & \sum_{i=1}^n \alpha_i y_i = 0. \end{aligned}$$

This dual depends only on inner products $x_i^\top x_j$, enabling the kernel trick by replacing them with $k(x_i, x_j)$. The optimal weights follow from the KKT conditions: only training points with $\alpha_i^\star > 0$ (the *support vectors*) contribute to $w^\star = \sum_i \alpha_i^\star y_i x_i$. Complementary slackness enforces $y_i(w^{\star\top} x_i + w_0^\star) = 1$ for active support vectors, which can be used to recover $w_0^\star$ by averaging over any $\alpha_i^\star > 0$.

Note. The dual formulation is preferable when n (number of samples) is smaller than d (feature dimension) or when kernels project data into high-dimensional spaces. It also highlights that margin maximization depends only on a sparse subset of training examples.

9.3 Kernelized SVMs

Linear SVMs can be extended by mapping inputs into a feature space $\phi(x)$ and learning a linear separator there. The kernel trick replaces inner products with a kernel function $k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$, so the dual becomes

$$\max_\alpha \quad \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

subject to $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$. The decision function depends only on kernel evaluations:

$$f(x) = \sum_{i=1}^n \alpha_i^\star y_i k(x_i, x) + w_0^\star, \quad \hat{y} = \operatorname{sign}(f(x)).$$

Common kernels include linear ($x^\top z$), polynomial ($(x^\top z + c)^p$), and RBF ($\exp(-\|x - z\|^2/(2\sigma^2))$). Valid kernels must correspond to a positive semidefinite Gram matrix (Mercer's condition).

9.4 Soft-margin SVMs

When the data are not perfectly separable, we introduce slack variables $\xi_i \geq 0$ to allow margin violations and penalize them in the objective:

$$\begin{aligned} \min_{w, w_0, \xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & y_i(w^\top x_i + w_0) \geq 1 - \xi_i, \quad i = 1, \ldots, n. \end{aligned}$$

Intuitively, each ξ_i measures how much sample i violates the margin: $\xi_i = 0$ is on or outside the margin, $0 < \xi_i < 1$ is inside the margin but correctly classified, and $\xi_i \geq 1$ is misclassified. The parameter C trades off margin width against violations: large C penalizes errors heavily (closer to hard margin), while small C allows more violations to achieve a wider margin and better generalization. The parameter C controls the trade-off between a wide margin and fewer violations. The dual is the same as the hard-margin case but with box constraints:

$$\max_{\alpha} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^\top x_j \quad \text{s.t.} \quad 0 \le \alpha_i \le C, \ \sum_i \alpha_i y_i = 0.$$

Soft-margin SVMs correspond to minimizing the hinge loss $\max(0, 1 - y_i f(x_i))$ plus $\frac{1}{2}\|w\|^2$ regularization.

9.5 Multiclass SVMs

For K classes, a linear multiclass SVM learns one weight vector per class, $w_1, \dots, w_K$, with scores $s_k(x) = w_k^\top x + w_{k,0}$. Prediction uses

$$\hat{y} = \arg \max_{k \in \{1,\dots,K\}} s_k(x).$$

The Crammer–Singer margin requires the correct class to outscore all others by at least m:

$$(w_{y_i}^\top x_i + w_{y_i,0}) - \max_{k \ne y_i}(w_k^\top x_i + w_{k,0}) \ge m, \quad \forall i.$$

This enforces a gap between the true class score and the best competing class, generalizing the binary margin to K-way discrimination. The hard-margin formulation is

$$\min_{\{w_k, w_{k,0}\}} \frac{1}{2}\sum_{k=1}^{K} \|w_k\|^2 \quad \text{s.t. the margin constraints above. (with } m = 1)$$

For non-separable data, introduce slacks $\xi_i \ge 0$:

$$\begin{aligned} \min_{\{w_k, w_{k,0}\}, \xi} \quad & \frac{1}{2}\sum_{k=1}^{K} \|w_k\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & (w_{y_i}^\top x_i + w_{y_i,0}) - \max_{k \ne y_i}(w_k^\top x_i + w_{k,0}) \ge 1 - \xi_i. \end{aligned}$$

The slack ξ_i measures how much the best incorrect class score overtakes the required margin; $\xi_i = 0$ means the example is correctly classified with margin, while $\xi_i > 0$ indicates a margin violation or misclassification. This is the multiclass analogue of the hinge-loss SVM and can be viewed as a special case of structured prediction with $\mathcal{K} = \{1, \dots, K\}$.

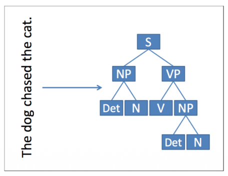
An example application is phoneme classification, where each audio frame must be assigned one of several phoneme labels. The multiclass SVM learns to discriminate among all phonemes simultaneously, maximizing the margin between the correct phoneme and the most confusable alternatives.

9.6 Structured SVMs

Structured SVMs extend SVMs to structured outputs $z \in \mathcal{K}$ (e.g., sequences, trees, segmentations). An example is part-of-speech tagging.

Four key problems to overcome for structured prediction are:

- **Compact output representation.** Even one parameter per structured label is infeasible; we need shared representations to avoid a parameter blow-up.
- **Efficient prediction.** Enumerating all outputs is intractable, so inference must exploit structure for fast maximization.
- **Prediction error.** Losses must reflect partial correctness (e.g., a nearly correct parse should be penalized less than a completely wrong one).
- **Efficient training.** Optimization must avoid constraints over all outputs, using methods with runtime sub-linear in the number of classes.



- S . . . sentence
- NP . . . noun phrase (*the dog, the cat*)
- *VP . . . verbal phrase (chased the cat)*
- *N . . . noun (dog, cat)*
- *V . . . verb (chased)*
- *Det . . . determiners (the)*

Figure 2: Structured SVM: language parsing example

The structured SVM framework generalizes binary and multiclass SVMs by replacing binary labels with a joint feature map $\Psi(z, y)$ and score function

$$f_w(z, y) = w^\top \Psi(z, y).$$

so the number of features depends on the dimensionality of the joint feature map only and is "independent" of the number of classes. Prediction is

$$\hat{z} = \arg\max_{z \in \mathcal{K}} f_w(z, y),$$

which requires efficient inference over the structured output space.

For training pairs (y_i, z_i), the hard-margin formulation enforces a structured margin:

$$w^\top \Psi(z_i, y_i) - \max_{z \neq z_i} w^\top \Psi(z, y_i) \geq 1, \quad \forall i.$$

This definition yields the optimization problem for hard functional margin SSVMs:

$$\begin{array}{ll} \min_{\mathbf{w}} & \frac{1}{2}\mathbf{w}^\top \mathbf{w} \\ \text{s.t.} & \mathbf{w}^\top \boldsymbol{\Psi}(z_i, \mathbf{y}_i) - \max_{z \neq z_i} \mathbf{w}^\top \boldsymbol{\Psi}(z, \mathbf{y}_i) \geq 1 \quad \forall \mathbf{y}_i \in \mathcal{Y} \end{array}$$

Classification requires computing

$$\hat{z} := h(\mathbf{y}) = \arg\max_{z \in \mathbb{K}} f_{\mathbf{w}}(z, \mathbf{y})$$

This is in general a hard combinatorial problem. For efficient classification, there must be some kind of structural matching between the compositional structure of the outputs z and the designed joint feature map Ψ. For example:

- Decomposable output spaces: The output space $\mathbb{K}$ can be decomposed into non-overlapping independent parts s.t. $\mathbb{K} = \mathbb{K}_1 \times \ldots \times \mathbb{K}_m$ (and Ψ respects this decomposition), then maximization can be performed part-wise.
- Specific dependency structures: A more general case is captured by Markov networks. Let z be a vector of random variables and $\boldsymbol{\Psi}(z, \mathbf{y})$ represent sufficient statistics of a conditional exponential model $P(z \mid \mathbf{y})$. Then, maximizing $f_{\mathbf{w}}(z, \mathbf{y})$ corresponds to finding the most probable output $\arg\max_z P(z \mid \mathbf{y})$. Fast inference methods available (e.g. Junction tree algorithm, Viterbi algorithm), depending on the dependency structure of z.

When data are not separable, introduce slacks and a task loss $\Delta(z, z_i)$ to rescale the margin (margin rescaling):

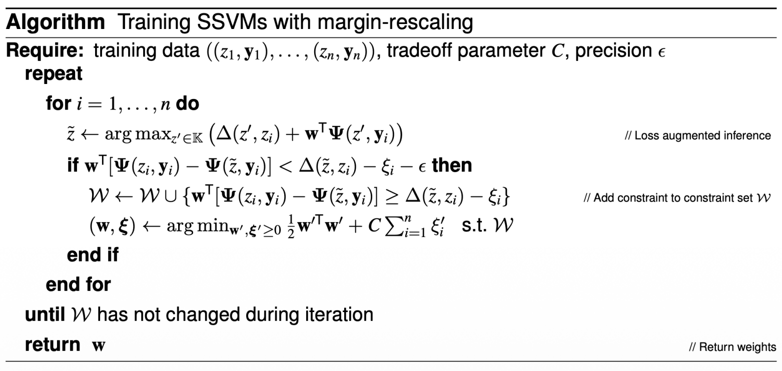
$$\begin{aligned} \min_{w, \xi \geq 0} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & w^\top \Psi(z_i, y_i) - w^\top \Psi(z, y_i) \geq \Delta(z, z_i) - \xi_i, \; \forall i, \; \forall z \neq z_i. \end{aligned}$$

Equivalently,

$$w^\top \Psi(z_i, y_i) - \max_{z \neq z_i} \big[\Delta(z, z_i) + w^\top \Psi(z, y_i)\big] \geq -\xi_i.$$

Training typically uses a cutting-plane algorithm that repeatedly performs loss-augmented inference

$$\tilde{z} = \arg\max_{z \in \mathcal{K}} \Delta(z, z_i) + w^\top \Psi(z, y_i)$$



```
Algorithm Training SSVMs with margin-rescaling
Require: training data $((z_1, \mathbf{y}_1), \dots, (z_n, \mathbf{y}_n))$, tradeoff parameter $C$, precision $\epsilon$
  repeat
    for $i = 1, \dots, n$ do
      $\tilde{z} \leftarrow \arg\max_{z' \in \mathbb{K}} \left( \Delta(z', z_i) + \mathbf{w}^\mathsf{T} \mathbf{\Psi}(z', \mathbf{y}_i) \right)$    // Loss augmented inference
      if $\mathbf{w}^\mathsf{T}[\mathbf{\Psi}(z_i, \mathbf{y}_i) - \mathbf{\Psi}(\tilde{z}, \mathbf{y}_i)] < \Delta(\tilde{z}, z_i) - \xi_i - \epsilon$ then
        $\mathcal{W} \leftarrow \mathcal{W} \cup \{\mathbf{w}^\mathsf{T}[\mathbf{\Psi}(z_i, \mathbf{y}_i) - \mathbf{\Psi}(\tilde{z}, \mathbf{y}_i)] \geq \Delta(\tilde{z}, z_i) - \xi_i\}$    // Add constraint to constraint set $\mathcal{W}$
        $(\mathbf{w}, \boldsymbol{\xi}) \leftarrow \arg\min_{\mathbf{w}', \boldsymbol{\xi}' \geq 0} \frac{1}{2} \mathbf{w}'^\mathsf{T} \mathbf{w}' + C \sum_{i=1}^{n} \xi'_i$  s.t. $\mathcal{W}$
      end if
    end for
  until $\mathcal{W}$ has not changed during iteration
  return $\mathbf{w}$    // Return weights
```

Figure 3: Structured SVM Training with Margin Rescaling

to add the most violated constraint. Designing Ψ, Δ, and efficient inference are the core modeling choices in structured prediction.

So, to apply structures SVMs to a new task we need to design/implement the following four functions:

- A joint feature map $\Psi(z, y)$ that captures relevant relationships between inputs and structured outputs.
- A loss function $\Delta(z, z_i)$ that quantifies the cost of predicting z when the true output is z_i.
- Loss augmented inference to efficiently solve $\arg\max_{z \in \mathcal{K}} \Delta(z, z_i) + w^\top \Psi(z, y_i)$ during training.
- Prediction rule to efficiently solve $\arg\max_{z \in \mathcal{K}} w^\top \Psi(z, y)$ at test time.

Note. **Privacy note.** SVM models can leak information about the data they were trained on because support vectors are stored implicitly (and sometimes explicitly) in the learned model. If test or sensitive data are inadvertently included during training, those examples can be memorized and potentially exposed through model inspection or membership inference.

10 Neural Networks

Neural networks are parameterized function families that represent a predictor as a composition of simple building blocks. Each layer applies an affine map followed by a nonlinearity. Stacking many such layers yields a flexible function that can model complicated input–output relationships while still being differentiable, which makes gradient-based training possible. The structure of this chapter is therefore: (i) define the forward computation, (ii) specify a loss and optimization objective, and (iii) derive how gradients flow backward through the composition.

10.1 Forward computation and parameterization

Let the network depth be L and define the parameters $\theta = \{W_i, b_i\}_{i=0}^{L-1}$. For an input $x \in \mathcal{X}$, we write the forward pass as

$$z_0 = x, \qquad \alpha_i = W_i z_i + b_i, \qquad z_{i+1} = \phi_i(\alpha_i), \quad i = 0, \dots, L-1.$$

The output is $z_L = f(x \mid \theta) \in \mathcal{Y}$. If the i-th layer has width d_i, then $W_i \in \mathbb{R}^{d_{i+1} \times d_i}$ and $b_i \in \mathbb{R}^{d_{i+1}}$. The representation z_i is often called a *hidden state* for $i \in \{1, \dots, L-1\}$.

The core intuition is that each layer performs a linear projection followed by a nonlinearity. The linear part mixes features; the nonlinearity allows the model to bend space and introduce interactions between features. Depth composes these distortions, creating increasingly abstract representations as information flows toward the output.

Note. If all ϕ_i were linear, then $f(x \mid \theta)$ would collapse to a single linear map, regardless of depth. Nonlinear activations are therefore essential for expressive power.

10.2 Activation functions and intuition

An activation function ϕ is typically applied elementwise and is almost everywhere differentiable. Common choices include

$$\text{sigmoid: } \sigma(a) = \frac{1}{1+e^{-a}}, \qquad \text{tanh: } \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}, \qquad \text{ReLU: } \mathrm{ReLU}(a) = \max(0, a).$$

Sigmoid and tanh saturate for large $|a|$, which can slow learning because their derivatives become small. ReLU is piecewise linear and keeps gradients alive for positive inputs, which helps optimization in deep networks. The choice of ϕ shapes both the representational geometry and the optimization landscape.

10.3 Loss, objective, and gradient descent

Given training data $D = \{(x_k, y_k)\}_{k=1}^{n}$, learning is posed as minimizing an empirical risk. Let $\ell(\hat{y}, y)$ be a differentiable loss for a single example (e.g., squared error for regression or cross-entropy for classification). The objective is

$$L(\theta, D) = \frac{1}{n} \sum_{k=1}^{n} \ell\big(f(x_k \mid \theta), y_k\big).$$

Gradient descent updates parameters in the direction of steepest decrease:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta, D),$$

with step size (learning rate) $\eta > 0$. In practice, one often uses stochastic or mini-batch variants, but the core algorithm remains the same: repeatedly evaluate gradients and take steps that lower the loss.

10.4 The chain rule as the engine of backpropagation

Training requires computing gradients of the loss with respect to all parameters. Because the network is a composition of functions, the chain rule gives a systematic way to propagate derivatives.

Consider differentiable functions $f : \mathbb{R}^d \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^n$, with $h = g \circ f$. Let $y = f(x)$ and $z = g(y)$. The Jacobian chain rule states

$$\frac{\partial h}{\partial x} = \frac{\partial g}{\partial y}\frac{\partial f}{\partial x}.$$

In a computation graph (a directed acyclic graph of intermediate variables), the derivative between two nodes is the sum, over all paths, of the product of local derivatives along each path. This is the mathematical justification for backpropagation: derivatives can be accumulated by traversing the graph backward.

$$\frac{\partial x'}{\partial x} = \sum_{(x_0,\ldots,x_m)\in\mathcal{P}(x,x')} \prod_{i=1}^{m} \frac{\partial x_i}{\partial x_{i-1}},$$

where $\mathcal{P}(x, x')$ is the set of all directed paths from x to x' in the graph. Each path contributes a product of local sensitivities; the total derivative is their sum.h

10.5 Backpropagation for a feed-forward network

We now apply the chain rule to the layered structure. For a single example, let

$$\mathcal{L}(x, y; \theta) = \ell(z_L, y)$$

be the loss. Define the *error signal* at layer i as

$$\delta_i = \frac{\partial \mathcal{L}}{\partial \alpha_i} \in \mathbb{R}^{d_{i+1}}.$$

The output layer error is

$$\delta_{L-1} = \nabla_{z_L}\ell(z_L, y) \odot \phi'_{L-1}(\alpha_{L-1}),$$

and for hidden layers, the error propagates backward as

$$\delta_i = (W_{i+1}^\top \delta_{i+1}) \odot \phi'_i(\alpha_i), \qquad i = L-2, \ldots, 0,$$

where $\odot$ is elementwise multiplication. Once the δ_i are known, the parameter gradients follow from the linear structure of each layer:

$$\frac{\partial \mathcal{L}}{\partial W_i} = \delta_i z_i^\top, \qquad \frac{\partial \mathcal{L}}{\partial b_i} = \delta_i.$$

Intuitively, δ_i measures how much changing the pre-activation α_i would change the loss. The gradient for W_i is then the outer product of this error with the input to the layer, z_i, which mirrors the forward computation $\alpha_i = W_i z_i + b_i$.

Derivation. For the layer $\alpha_i = W_i z_i + b_i$ and $z_{i+1} = \phi_i(\alpha_i)$, consider a scalar loss $\mathcal{L}$. By the chain rule,

$$\frac{\partial \mathcal{L}}{\partial W_i} = \frac{\partial \mathcal{L}}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial W_i}.$$

The derivative of α_i with respect to W_i is linear: each entry of W_i affects α_i in proportion to the corresponding entry of z_i. In matrix form, this gives

$$\frac{\partial \mathcal{L}}{\partial W_i} = \delta_i z_i^\top.$$

Similarly, because α_i depends on b_i by simple addition, we obtain $\partial\mathcal{L}/\partial b_i = \delta_i$.

Backpropagation therefore consists of two sweeps: a forward pass that computes α_i and z_i for all layers, and a backward pass that computes δ_i and parameter gradients. This is efficient because each intermediate quantity is reused, and the total cost is comparable to the forward pass itself.

Note. The same derivation applies to the full dataset: one either sums gradients over all examples (batch) or estimates them from a subset (mini-batch). In all cases, the gradient structure follows the same backward recursion.

11 Transformers

Transformers are sequence models built around attention. Instead of processing tokens strictly left-to-right, they let every token form a context-aware representation by looking at other tokens. The narrative of the chapter is: (i) represent text as token embeddings, (ii) learn how tokens attend to each other via self-attention, (iii) extend attention to multiple relations and to encoder–decoder settings, and (iv) add position and depth to form the full transformer architecture.

11.1 Tokenization and embeddings

Text must be converted into discrete tokens. A common approach is WordPiece tokenization, which builds a vocabulary of size S by merging frequent character pairs:

1. Initialize the token set with all characters appearing in the corpus.
2. While the set size exceeds S, repeatedly merge the most frequent adjacent token pair.

Each token is mapped to a learnable embedding in $\mathbb{R}^d$. Stacking all token vectors yields an embedding matrix $E \in \mathbb{R}^{S\times d}$. For a token sequence $t_1, \ldots, t_N$, the model selects the corresponding rows of E to build a matrix $X \in \mathbb{R}^{N\times d}$. In classification tasks, a special token like `[CLS]` is added so its final embedding can represent the whole sequence.

11.2 Self-attention: context for each token

Self-attention builds a new representation for each token by mixing information from all other tokens. The mechanism assigns a weight to every pair of positions and uses these weights to take a weighted average of value vectors.

Given input embeddings $X \in \mathbb{R}^{N\times d}$, define

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V,$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d\times d_k}$. The attention weights are

$$S = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right),$$

with softmax applied row-wise, and the attention output is

$$A = SV.$$

The (i, j) entry of S measures how much token i attends to token j. Each output row A_i is therefore a weighted average of value vectors, which means each token representation is rewritten using a context-dependent mixture of other tokens.

Note. The scaling by $\sqrt{d_k}$ keeps dot products in a reasonable range. Without it, $QK^\top$ grows in magnitude with dimension, causing the softmax to saturate and gradients to vanish. The scaling normalizes the variance of the scores so attention remains learnable.

11.3 Attention as information retrieval

Attention can be interpreted as a retrieval mechanism. Each token produces a query vector that asks, *which other tokens are relevant to me?* Keys describe how each token can be matched, and values are the information retrieved. The softmax turns the query–key similarities into a probability distribution, so the output is a weighted average of values. This perspective explains why attention helps disambiguate word meaning: a token like *bat* can retrieve different context depending on whether nearby words indicate sports or animals.

11.4 Multi-head attention

A single attention map captures one kind of relation, but language contains many relations at once (subject–verb, modifier–noun, long-range coreference). Multi-head attention learns several attention maps in parallel. For head h,

$$Q_h = XW_Q^{(h)}, \quad K_h = XW_K^{(h)}, \quad V_h = XW_V^{(h)}, \quad A_h = \mathrm{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}}\right) V_h.$$

The head outputs are concatenated and projected:

$$\mathrm{MHA}(X) = \mathrm{Concat}(A_1, \ldots, A_H) W_O.$$

The intuition is that different heads specialize to different relational patterns, and concatenation preserves these diverse views before the final mixing step.

11.5 Cross-attention

In tasks like translation, an output token should attend not only to previous output tokens but also to the input sentence. Cross-attention does this by forming queries from the target sequence and keys/values from the source sequence. If X_s are source embeddings and X_t are target embeddings, then

$$Q = X_t W_Q, \qquad K = X_s W_K, \qquad V = X_s W_V,$$

so each target position retrieves information from the source. This gives a direct, learnable alignment between the two sequences.

11.6 Masked self-attention for autoregressive decoding

When generating text left-to-right, a token must not look at future positions. This is enforced by a causal mask $M \in \{0,1\}^{N\times N}$ that zeroes out disallowed positions. If $P = QK^\top/\sqrt{d_k}$ are the raw scores, we apply

$$P_M = \mu(P, M), \qquad \mu(p, m) = \begin{cases} p & m = 1 \\ -\infty & m = 0 \end{cases}$$

and then compute $S = \mathrm{softmax}(P_M)$. The $-\infty$ entries become zeros after softmax, ensuring each position only attends to the past. This preserves the autoregressive factorization while keeping the attention computation parallelizable across positions.

11.7 Positional encodings

Self-attention alone is permutation equivariant: reordering tokens simply reorders the outputs. To inject order, a positional encoding matrix $P \in \mathbb{R}^{T\times d}$ is added to the token embeddings, where T is the maximum sequence length. A common deterministic scheme uses sinusoids:

$$P_{i,2j} = \sin\left(i \cdot \alpha^{-2j}\right), \qquad P_{i,2j+1} = \cos\left(i \cdot \alpha^{-2j}\right), \qquad \alpha = 10^{4/d}.$$

These features provide multiple periodicities, allowing the model to represent both absolute and relative positions. Adding P to token embeddings keeps the dimensionality fixed; concatenation would increase parameters and would be less compatible with the linear structure of attention and feed-forward layers.

11.8 Residual connections and normalization

Deep networks can suffer from degradation: accuracy drops as depth grows because information and gradients struggle to pass through many layers. Transformers address this by using residual connections and layer normalization. Each sublayer is wrapped as

$$\mathrm{AddNorm}(x) = \mathrm{LayerNorm}(x + \mathrm{Sublayer}(x)).$$

The residual path preserves the original signal, while normalization stabilizes the scale of activations. Together, they enable very deep stacks of attention and feed-forward blocks.

11.9 The transformer block and full architecture

An encoder block consists of multi-head self-attention followed by a position-wise feed-forward network (FFN):

$$\mathrm{FFN}(z) = W_2\sigma(W_1 z + b_1) + b_2,$$

applied independently to each position. The encoder stacks L such blocks.

A decoder block contains masked self-attention, cross-attention to the encoder output, and an FFN, each with Add&Norm. The final decoder representations are mapped to vocabulary logits by a linear layer and softmax, producing a distribution over the next token.

This architecture combines parallelizable self-attention with explicit alignment via cross-attention, making it effective for translation, summarization, and many other sequence tasks.

11.10 BERT as an encoder-only transformer

BERT uses only the encoder stack. It is pre-trained on large corpora with two objectives:

- **Masked language modeling (MLM):** randomly mask input tokens and predict them from context.
- **Next sentence prediction (NSP):** classify whether one sentence follows another.

After pre-training, the model is fine-tuned for downstream tasks such as question answering or sentence classification. The key intuition is that bidirectional self-attention yields contextual embeddings that can be specialized with minimal task-specific changes.

12 Diffusion

Use the PDF from moodle for this chapter.

13 Graph Neural Networks

Graphs encode relational data: molecules are atoms connected by bonds, social networks are people connected by interactions, and knowledge bases are entities linked by facts. A graph is a pair $G = (V, E)$ with vertices V and edges $E \subseteq V \times V$. In many learning problems, vertices (and sometimes edges) carry features. We will focus on node-annotated graphs, where each vertex $v \in V$ has a feature vector $h_v \in \mathbb{R}^d$, and on undirected graphs for clarity. The main goal is to learn a function f^* that maps graphs to targets (graph-level prediction) or to labels per node (node-level prediction), while respecting the symmetries of graphs.

13.1 Graphs, annotations, and learning objectives

Let p^* be a distribution over annotated graphs and let f^* be an unknown target function. We want a model f such that $f(G)$ approximates $f^*(G)$ for $G \sim p^*$. A core constraint is *permutation invariance*: reordering vertices should not change the graph-level prediction, and should only reorder the node-level outputs. Any graph neural network (GNN) must therefore aggregate information in a way that ignores the arbitrary indexing of nodes.

13.2 From layers to graph filters

Standard neural networks repeatedly apply a linear map followed by a nonlinearity, and convolutional networks do the same but on local image patches. Graphs do not have a fixed grid, so the local patch around a node must be defined by connectivity. The natural choice is the node and its neighbors.

Let $N(u)$ be the neighbors of node u and let $\deg(u)$ be its degree. A widely used graph convolutional update is

$$h_u^{(\ell+1)} = \phi\left(\frac{1}{\sqrt{1+\deg(u)}} \sum_{v \in N(u) \cup \{u\}} \frac{1}{\sqrt{1+\deg(v)}} h_v^{(\ell)} W^{(\ell)}\right),$$

where $h_v^{(\ell)}$ is the feature vector (hidden state) of node v at layer ℓ, $W^{(\ell)}$ is a learnable matrix, and ϕ is an elementwise activation. The symmetric scaling by $\sqrt{1+\deg(\cdot)}$ keeps feature magnitudes comparable across nodes with different degrees.

The two degree factors can be read as symmetric normalization. Scaling by $1/\sqrt{1+\deg(v)}$ limits how much a high-degree sender can contribute to many neighbors, while $1/\sqrt{1+\deg(u)}$ prevents a high-degree receiver from accumulating excessive mass. Together they yield the normalized operator $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, which is symmetric on undirected graphs, keeps eigenvalues in a stable range, and makes repeated message passing behave like a degree-aware averaging rather than an explosion or collapse driven by hubs.

In matrix form, stack node features as rows in $H^{(\ell)} \in \mathbb{R}^{n \times d}$, add self-loops to the adjacency matrix A by defining $\tilde{A} = A + I$, and let $\tilde{D}$ be the diagonal degree matrix with $\tilde{D}_{ii} = 1 + \deg(i)$ (degree in $\tilde{A}$). Then

$$H^{(\ell+1)} = \phi\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} H^{(\ell)} W^{(\ell)}\right).$$

The operator $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ performs a normalized neighbor average, analogous to a convolutional filter on an irregular domain.

13.3 Message passing and expressive power

The update above is a special case of a *message passing* scheme. At each layer, a node sends a message derived from its current state, receives messages from neighbors, aggregates them (often by sum), and updates its representation. This mirrors distributed graph algorithms, where local communication and computation are iterated to compute global properties.

This connection explains why GNNs are expressive. Message passing algorithms can implement procedures such as breadth-first search or shortest paths, and GNNs can be seen as differentiable, learnable versions of such algorithms. Formal results show that message passing GNNs are universal approximators of graph functions when given sufficient width and depth: by stacking enough layers and choosing appropriate nonlinear transformations, they can approximate any continuous function that respects graph symmetry.

13.4 Oversmoothing as a spectral phenomenon

Depth brings a challenge: repeated neighbor averaging can make node representations indistinguishable, a phenomenon known as *oversmoothing.* To see why, define

$$S = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}.$$

If we temporarily ignore nonlinearities and weight matrices, the update reduces to $H^{(\ell)} = S^{\ell}X$, where X is the input feature matrix. The matrix S is symmetric with eigenvalues

$$1 = \lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n > -1.$$

Let $S = U\Lambda U^{\top}$ be its eigendecomposition. Then

$$H^{(\ell)} = U\Lambda^{\ell}U^{\top}X \to u_1u_1^{\top}X \quad \text{as } \ell \to \infty,$$

where u_1 is the eigenvector of the largest eigenvalue. This means all node features become aligned with u_1: the representations collapse into a one-dimensional subspace. The rate of collapse is exponential,

$$\|S^{\ell} - u_1u_1^{\top}\|_2 = \max_{i\geq 2} |\lambda_i|^{\ell},$$

so deeper layers quickly erase distinctions between nodes. Intuitively, repeated averaging acts as a low-pass filter on the graph: high-frequency variations across neighbors vanish, and all nodes converge to a similar state.

13.5 Mitigating oversmoothing

Graph Isomorphism Networks (GIN). Oversmoothing is tied to averaging. One way to reduce it is to use sum aggregation without degree normalization and then apply a powerful MLP:

$$h_v^{(\ell+1)} = \mathrm{MLP}^{(\ell)}\left((1+\epsilon^{(\ell)})h_v^{(\ell)} + \sum_{u\in N(v)} h_u^{(\ell)}\right).$$

The sum preserves the multiset of neighbor features, and the MLP can learn a rich, injective transformation. This update can yield eigenvalues with magnitude greater than 1 in the effective propagation operator, counteracting the exponential shrinkage that causes oversmoothing.

Laplace positional encodings. Another strategy is to augment node features with *spectral coordinates* that resist smoothing. Consider the eigenvectors of S (or, equivalently, of the normalized graph Laplacian). The top eigenvector u_1 is the smoothest signal and dominates the limit of S^{ℓ}, so we instead take the next k eigenvectors $u_2, \ldots, u_{k+1}$ and append them to the input features:

$$X' = [X \mid u_2 \mid \cdots \mid u_{k+1}].$$

These eigenvectors are orthogonal to u_1 and correspond to directions that decay more slowly under S^{ℓ}. As a result, they provide positional information that helps the model preserve structural differences across nodes even in deeper networks.

14 Anomaly Detection

The task of anomaly detection involves identifying data points that deviate significantly from the norm. In situations like banks needing to detect fraudulent transactions, or hospitals aiming to identify unusual patient health metrics, effective anomaly detection methods are crucial.

14.1 Anomalies

Typically, we have a space of all possible events $\mathcal{X}$ and a subset called the normal set $\mathcal{N} \subseteq \mathcal{X}$. An anomaly is any event $x \in \mathcal{X}$ such that $x \notin \mathcal{N}$. What makes this challenging is that the normal set $\mathcal{N}$ is often not explicitly defined, and we may only have access to a limited set of examples from $\mathcal{N}$. The way we approach this problem is to do a dimensionality reduction Π on the data to find a lower-dimensional representation that captures the essential structure of the normal data. We can then use this representation to identify anomalies.

The generalized approach looks like this:

1. An anomaly is an unlikely event.
2. We fit a model of a parametric family of distributions $\mathcal{H} = \{p(x;\theta) : \theta \in \Theta\}$
3. We define an anomaly score $-\log p_{\hat{\theta}}(x)$

We choose a GMM (Gaussian Mixture Model) as our parametric family of distributions. A GMM is a weighted sum of multiple Gaussian distributions, which allows us to model complex data distributions effectively. The parameters θ of the GMM include the means, covariances, and mixture weights of the individual Gaussian components. This is a good choice, because it has been observed that linear projections of high-dimensional distributions onto low-dimensional spaces resemble Gaussian distributions, thie can partially be explained by the Central Limit Theorem.

This leads us to the following algorithm for anomaly detection using PCA and GMMs:

1. Project the data linearly onto a lower-dimensional space (using PCA)
2. Fit a GMM to the projected data

14.2 Dimensionality Reduction

Given $X = \{x_1, \dots, x_n\} \subseteq \mathbb{R}^D$, find a linear projection $\pi : \mathbb{R}^D \to \mathbb{R}^d$, with $d \ll D$, and such that $\pi(X)$ has a "sufficiently large" variance. So we want a projection π that maximizes the variance of the projected data points.

14.2.1 First Stage

We start with a very tractable case which might be a bit too simple. We look for a one-dimensional projection $\pi(x) = w^T x$, where $w \in \mathbb{R}^D$ is a unit vector (i.e., $\|w\|_2 = 1$). The variance of the projected data points is given by:

$$\mathrm{Var}(\pi(X)) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - w^T \mu)^2 = w^T \Sigma w \tag{4}$$

where $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the mean of the data points, and $\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$ is the covariance matrix of the data points. To find the optimal projection vector w, we need to solve the following optimization problem:

$$\max_{w \in \mathbb{R}^D} w^T \Sigma w \quad \text{subject to} \quad \|w\|_2 = 1 \tag{5}$$

Where the solution is given by the eigenvector of Σ corresponding to the largest eigenvalue.

14.2.2 General Case

To generalize this to a d-dimensional projection, we can define the projection as $\pi(x) = W^T x$, where $W \in \mathbb{R}^{D\times d}$ is a matrix with orthonormal columns (i.e., $W^T W = I_d$). We proceed similarly to the one-dimensional case, and we want to maximize the variance of the projected data points:

$$\operatorname{Var}(\pi(X)) = \frac{1}{n}\sum_{i=1}^{n} \|W^T x_i - W^T \mu\|_2^2 = \operatorname{tr}(W^T \Sigma W) \tag{6}$$

To find the optimal projection matrix W, we need to solve the following optimization problem:

$$\max_{W\in\mathbb{R}^{D\times d}} \operatorname{tr}(W^T \Sigma W) \quad \text{subject to} \quad W^T W = I_d \tag{7}$$

The solution is given by the matrix W whose columns are the eigenvectors of Σ corresponding to the d largest eigenvalues.

14.3 Fitting a GMM

14.3.1 GMM Definition

A Gaussian Mixture Model (GMM) is a probabilistic model that assumes that the data is generated from a mixture of several Gaussian distributions. The probability density function of a GMM with K components is given by:

$$p(x;\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x;\mu_k,\Sigma_k) \tag{8}$$

where π_k are the mixture weights (with $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$), μ_k are the means, and Σ_k are the covariance matrices of the individual Gaussian components. The parameters of the GMM are denoted by $\theta = \{\pi_k,\mu_k,\Sigma_k\}_{k=1}^{K}$.

14.4 Fitting

We again use a MLE approach, that is, we want to find the parameters $\hat{\theta}$ that maximize the likelihood of the observed data:

$$\begin{aligned}
\log p_\theta(x) &= \log \prod_{i=1}^{n} p(x_i;\theta) &(9)\\
&= \sum_{i=1}^{n} \log p(x_i;\theta) &(10)\\
&= \sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k,\Sigma_k)\right) &(11)
\end{aligned}$$

However, this last term is difficult to optimize directly due to the presence of the logarithm of a sum. We make an interesting observation: if we knew which component generated each data point, the optimization would be much simpler. This leads us to introduce latent variables z_i that indicate the component responsible for generating each data point x_i. Specifically, we define z_i as a one-hot encoded vector where $z_{ik} = 1$ if the i-th data point was generated by the k-th component, and 0 otherwise. With these latent variables, we can express the complete-data log-likelihood as:

$$\begin{aligned}
\log p_\theta(X,Z) &= \log p_\theta(Z) p_\theta(X \mid Z) &(12)\\
&= \log p_\theta(Z) + \log p_\theta(X \mid Z) &(13)\\
&= \sum_{i\leq n} \log \pi_{Z_i} + \sum_{i\leq n} \log p_\theta\left(X_i \mid Z_i\right) &(14)
\end{aligned}$$

Where $Z = \{z_1,\ldots,z_n\}$ is the set of latent variables for all data points. This is way easier to optimize.

Now we have the setup, that we have $\log p_\theta(x)$ but we would like to work with $\log p_\theta(X, Z)$. We know that:

$$\log p_\theta(x) = \log \frac{p_\theta(x, z)}{p_\theta(z|x)} \tag{15}$$

$$\Rightarrow \mathbb{E}_{Z\sim q}[\log p_\theta(x)] = \mathbb{E}_{Z\sim q}\left[\log \frac{p_\theta(x, Z)}{p_\theta(Z|x)}\right] \tag{16}$$

$$\tag{17}$$

Since the left-hand side does not depend on Z, we can write:

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{Z\sim q}\left[\log \frac{p_\theta(x, Z)}{p_\theta(Z|x)}\right] && (18)\\
&= \mathbb{E}_{Z\sim q}\left[\log \frac{p_\theta(x, Z)}{p_\theta(Z|x)} \cdot \frac{q(Z)}{q(Z)}\right] && (19)\\
&= \mathbb{E}_{Z\sim q}[\log p_\theta(x, Z) - \log q(Z)] + \mathbb{E}_{Z\sim q}\left[\log \frac{q(Z)}{p_\theta(Z|x)}\right] && (20)\\
&= M(q, \theta) + \mathrm{KL}(q \| p_\theta(Z|x)) && (21)
\end{aligned}$$

Where we defined:

$$M(q, \theta) = \mathbb{E}_{Z\sim q}[\log p_\theta(x, Z) - \log q(Z)] \tag{22}$$

This gives us a lower bound on the log-likelihood, since the KL divergence is always non-negative:

$$\log p_\theta(x) \geq M(q, \theta) \tag{23}$$

and we have equality if and only if $q(Z) = p_\theta(Z|x)$.

14.5 EM Algorithm

The Expectation-Maximization (EM) algorithm is an iterative method used to find maximum likelihood estimates of parameters in statistical models with latent variables. In the context of fitting a GMM, the EM algorithm alternates between two main steps: the Expectation (E) step and the Maximization (M) step.

- **E-step:** In this step, we compute the expected value of the complete-data log-likelihood with respect to the current estimate of the parameters $\theta^{(t)}$. This involves calculating the posterior probabilities of the latent variables given the observed data and the current parameter estimates. Specifically, we set:

$$q^{(t+1)}(Z) = p_{\theta^{(t)}}(Z|X) \tag{24}$$

- **M-step:** In this step, we maximize the expected complete-data log-likelihood computed in the E-step with respect to the parameters θ. This gives us updated parameter estimates:

$$\theta^{(t+1)} = \arg\max_\theta M(q^{(t+1)}, \theta) \tag{25}$$

The EM algorithm iterates between these two steps until convergence, which is typically determined by checking if the change in the log-likelihood or the parameter estimates falls below a predefined threshold.

14.6 Validation Metrics

We want a function $\phi : \mathbb{R}^D \to \{0, 1\}$ that that classifies points as normal (0) or anomalous (1). Given a threshold τ. We have two objectives: (let $C = \{x : \phi(x) = 1\}$ be the set of points classified as anomalous, and A be the set of true anomalies)

- If $x \in A$, then $\phi(x) = 1$

$$\text{Recall} = \frac{|C \cap A|}{|A|} \tag{26}$$

'Reatio of the correctly identified anomalous among **all truly anomalous** points'

How many of the actual anomalous points did we identify?

- If $\phi(x) = 1$, then $x \in A$ (Precision)

$$\text{Precision} = \frac{|C \cap A|}{|C|} \tag{27}$$

  'Ratio of the correctly identified anomalous points among **all identified anomalous points**'

  How many of the points we classified as anomalous are actually anomalous?

We can combine these two metrics into a single metric called the F1-score, which is the harmonic mean of precision and recall:

$$\text{F1 Score } = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \tag{28}$$

The F1-score provides a balanced measure of the model's performance in identifying anomalies, taking into account both precision and recall.

14.7 Combined Pipeline

Given a set $X \in \mathbb{R}^D$ of "normal" points, we train an anomaly detector as follows.

1. We compute a projector $\pi : \mathbb{R}^D \to \mathbb{R}^d$ using PCA.
2. We then fit a pdf $p_\theta(\cdot)$ with k -components to $\{\pi(x) : x \in X\}$, using the EM algorithm.
3. For a new point, its "anomaly score" is $-\log p_\theta(\pi(x))$.

15 Reinforcement Learning

Reinforcement learning addresses the problem of learning to make sequential decisions through interaction with an environment. Unlike supervised learning, where we are given labeled examples, in reinforcement learning an agent must discover which actions lead to desirable outcomes by trying them and observing rewards. This chapter presents the mathematical foundations of reinforcement learning, beginning with the Markov decision process formalism and developing the key concepts of value functions, Bellman equations, and learning algorithms.

15.1 Agent-Environment Interaction Loop

Reinforcement learning is organized around repeated interaction. At time step t, the agent observes a state s_t (often an observation o_t in practice), selects an action a_t, receives a reward r_t, and the environment transitions to s_{t+1}. Episodes terminate when a goal is reached or a failure condition occurs. A typical training loop alternates between:

- **Generate samples:** roll out the current policy to collect trajectories.
- **Estimate performance:** compute returns or advantages from the collected data.
- **Update policy:** adjust parameters to improve expected return.

In practical RL libraries (e.g., Gymnasium), the interaction is standardized via `reset()` to start an episode and `step(action)` to advance the environment until it terminates or is truncated. Tasks vary in whether states and actions are discrete or continuous; tabular methods can handle small discrete spaces, while continuous or high-dimensional settings require function approximation, typically with neural networks.

15.2 Markov Decision Processes

We formalize the reinforcement learning problem using the framework of Markov decision processes (MDPs), which provide a mathematical model for sequential decision-making under uncertainty.

Definition 1 (Markov Decision Process) *A Markov decision process is a tuple* $(\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$*, where* $\gamma \in (0, 1]$*, consisting of:*

- $\mathcal{S}$*: the state space (measurable space if continuous)*
- $\mathcal{A}$*: the action set (discrete or continuous)*
- $p(\cdot \mid s, a)$*: the transition kernel over* $\mathcal{S}$
- $r(s, a, s') \in \mathbb{R}$*: the reward function*
- ρ_0*: the initial-state distribution*
- γ*: the discount factor*

The state space $\mathcal{S}$ describes all possible configurations of the environment. At each time step, the agent observes a state $s \in \mathcal{S}$, selects an action $a \in \mathcal{A}$, and the environment transitions to a new state s' sampled from $p(\cdot \mid s, a)$, yielding reward $r(s, a, s')$.

The discount factor γ determines how the agent weighs future rewards relative to immediate rewards. When γ is close to 0, the agent behaves myopically, prioritizing immediate rewards. When γ approaches 1, future rewards are valued nearly as much as immediate ones. The discount factor also ensures that infinite sums of rewards remain bounded.

Note. In a partially observable MDP (POMDP), the agent does not observe the true state s directly. Instead, it perceives an observation $o \sim \mathcal{O}(\cdot \mid s)$ drawn from an observation distribution. In a fully observable MDP, $o = s$ (full observability). We focus on fully observable MDPs throughout this chapter.

15.3 Policies

A policy specifies the agent's behavior by mapping states to action distributions.

Definition 2 (Policy) *A policy is a function π that maps each state $s \in \mathcal{S}$ to a distribution over the action set $\pi(\cdot \mid s)$.*

We distinguish between two types of policies:

- **Deterministic**: $\mu : \mathcal{S} \to \mathcal{A}$, when $\pi(\cdot \mid s)$ has no randomness.
- **Stochastic**: $\pi(\cdot \mid s)$, otherwise.

A deterministic policy always selects the same action in a given state, while a stochastic policy may randomize. Stochastic policies are useful for exploration and, in some settings (such as games with simultaneous moves), may be necessary for optimality.

15.4 Trajectories and Returns

Given a policy π, the agent's interaction with the environment generates a trajectory:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots), \tag{29}$$

where $s_0 \sim \rho_0$, $a_t \sim \pi(\cdot \mid s_t)$, $s_{t+1} \sim p(\cdot \mid s_t, a_t)$, and $r_t = r(s_t, a_t, s_{t+1})$.

The discounted return from time t is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{30}$$

The return G_t represents the total discounted reward accumulated from time t onward. It is a random variable because future states, actions, and rewards are stochastic. The objective in reinforcement learning is to find a policy that maximizes the expected return.

An important property of the return is that it satisfies a recursive decomposition:

$$G_t = r_t + \gamma G_{t+1}. \tag{31}$$

This recursive structure is fundamental to the Bellman equations we will derive shortly.

15.5 The Markov Property

The MDP framework is built on the Markov property, which states that the next state depends only on the current state and action, not on the entire history:

$$\mathbb{P}(s_{t+1} \mid s_{0:t}, a_{0:t}) = \mathbb{P}(s_{t+1} \mid s_t, a_t). \tag{32}$$

This property has two important consequences:

- The current state-action pair (s_t, a_t) is a sufficient statistic for predicting the future. We do not need to track the entire history of states and actions.
- The Markov property enables dynamic programming and Bellman recursion. Because the future depends only on the present, we can express long-term values recursively in terms of immediate rewards and future values.

The Markov property is what makes many reinforcement learning algorithms tractable. Without it, we would need to condition on arbitrarily long histories, making the problem combinatorially complex.

15.6 Value Functions

To evaluate policies, we define value functions that measure the expected return from a given state or state-action pair.

Definition 3 (Value Functions) *For a policy π, we define:*

$$V^\pi(s) = \mathbb{E}_\pi[G \mid s], \tag{33}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi[G \mid s, a], \tag{34}$$

where the expectation is taken over trajectories generated by following policy π.

The state-value function $V^\pi(s)$ gives the expected return starting from state s and following policy π. The action-value function $Q^\pi(s,a)$ gives the expected return starting from state s, taking action a, and then following policy π.

For discrete action spaces, these two value functions are related by:

$$V^\pi(s) = \sum_a \pi(a \mid s) Q^\pi(s,a). \tag{35}$$

The intuition is simple: the value of a state under policy π is the expected Q-value of actions drawn from π. If we know all the Q-values, we can compute the state-value by averaging over the policy's action distribution.

15.7 Bellman Equations

The Bellman equations express value functions recursively, exploiting the Markov property to relate values at one time step to values at the next.

15.7.1 Policy Evaluation

For a fixed policy π, the value functions satisfy the following Bellman expectation equations:

$$V^\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s,a) \left(r(s,a,s') + \gamma V^\pi(s')\right), \tag{36}$$

$$Q^\pi(s,a) = \sum_{s'} p(s' \mid s,a) \left(r(s,a,s') + \gamma \sum_{a'} \pi(a' \mid s') Q^\pi(s',a') \right). \tag{37}$$

Derivation. We derive equation (36). Starting from the definition of V^π and using the recursive structure of the return:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \mathbb{E}_\pi[r_t + \gamma G_{t+1} \mid s_t = s] \\ &= \mathbb{E}_\pi[r_t \mid s_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} \mid s_t = s]. \end{aligned}$$

The first term involves averaging over actions drawn from $\pi(\cdot \mid s)$ and next states from $p(\cdot \mid s,a)$:

$$\mathbb{E}_\pi[r_t \mid s_t = s] = \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s,a) r(s,a,s').$$

For the second term, we use the law of total expectation and the Markov property:

$$\begin{aligned} \mathbb{E}_\pi[G_{t+1} \mid s_t = s] &= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s,a) \mathbb{E}_\pi[G_{t+1} \mid s_{t+1} = s'] \\ &= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s,a) V^\pi(s'). \end{aligned}$$

Combining these results yields equation (36). The derivation of equation (37) follows a similar structure.

These equations express the value of a state (or state-action pair) in terms of immediate rewards and the values of successor states. They form a system of linear equations that can be solved to find V^π or Q^π for a given policy π—a process called policy evaluation.

15.7.2 Optimality

Rather than evaluating a specific policy, we often seek an optimal policy that maximizes expected return from every state. The optimal value functions are:

$$V^*(s) = \max_a \sum_{s'} p(s' \mid s, a) \left(r(s, a, s') + \gamma V^*(s') \right), \tag{38}$$

$$Q^*(s, a) = \sum_{s'} p(s' \mid s, a) \left(r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right). \tag{39}$$

These are the Bellman optimality equations. Unlike the Bellman expectation equations, which average over the policy's action distribution, the optimality equations maximize over actions. The optimal value of a state is achieved by taking the best action from that state.

Once we have computed Q^*, we can extract an optimal policy by acting greedily:

$$\pi^*(s) = \arg\max_a Q^*(s, a). \tag{40}$$

Because $Q^*(s, a)$ already accounts for optimal future behavior, selecting the action with the highest Q-value in each state yields an optimal policy.

15.8 Exploration versus Exploitation

A central challenge in reinforcement learning is balancing exploration and exploitation. To learn accurate value estimates, the agent must explore different actions to discover their consequences. However, to maximize reward, the agent should exploit its current knowledge by selecting actions it believes to be best. These objectives are in conflict.

The goal is to maximize expected return while reducing uncertainty. Several strategies address this trade-off:

ϵ-Greedy. With probability ϵ, the agent selects a random action uniformly from $\mathcal{A}$. With probability $1 - \epsilon$, it acts greedily: $\arg\max_a Q(s, a)$. This ensures sufficient exploration while still exploiting good actions most of the time.

Softmax/Boltzmann. The agent samples actions according to a Boltzmann distribution:

$$\pi(a \mid s) \propto e^{Q(s,a)/\tau}, \tag{41}$$

where $\tau > 0$ is a temperature parameter. Large τ yields nearly uniform exploration, while small τ concentrates probability on the greedy action.

Annealing. Both ϵ and τ are typically annealed over time—gradually decreased as learning progresses. Early in training, high exploration helps discover good actions. Later, exploitation of learned knowledge becomes more valuable.

15.9 On-Policy versus Off-Policy Learning

Reinforcement learning algorithms can be categorized by how they relate the policy being learned to the policy generating experience. This distinction has fundamental implications for data efficiency and algorithm design.

Definition 4 (On-Policy and Off-Policy Learning)

- ***On-policy learning**: The agent learns about and improves the same policy π that it uses to generate experience. The policy being evaluated is the policy being executed.*

- ***Off-policy learning**: The agent learns about a target policy (often the optimal policy) while following a different behavior policy for exploration.*

The key difference lies in data reuse and exploration flexibility:

On-Policy Methods. In on-policy learning, the agent must use recent data generated by the current policy. After each policy update, the data collected under the old policy becomes stale and typically cannot be reused. This ensures that value estimates and gradient computations accurately reflect the current policy, but requires continuous fresh data collection. Examples include SARSA and basic policy gradient methods like REINFORCE.

Off-Policy Methods. Off-policy learning decouples the behavior policy (used for exploration) from the target policy (being learned). The agent can learn about the optimal policy while following an exploratory policy like ϵ-greedy. Crucially, off-policy methods can reuse old data from the replay buffer, improving sample efficiency. However, they require techniques like importance sampling or bootstrapping to correct for the mismatch between behavior and target policies. Q-learning is a canonical off-policy algorithm.

The distinction affects practical considerations:

- **Sample efficiency**: Off-policy methods can learn from any experience, including old data or demonstrations, making them more sample-efficient when experience is expensive.
- **Stability**: On-policy methods tend to be more stable because they avoid the distribution shift between behavior and target policies.
- **Parallelization**: On-policy methods can easily parallelize data collection across multiple workers, while off-policy methods can share a single replay buffer across multiple learners.

15.10 Q-Learning

Q-learning is a fundamental algorithm that learns the optimal action-value function Q^* from experience, without requiring a model of the environment's dynamics.

15.10.1 The Update Rule

Q-learning is an off-policy temporal-difference learning algorithm. After observing a transition (s_t, a_t, r_t, s_{t+1}), Q-learning updates the Q-value estimate:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right], \tag{42}$$

where α is the learning rate.

The quantity in brackets is the temporal-difference (TD) error. It measures the discrepancy between the current Q-value estimate $Q(s_t, a_t)$ and a target value $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$ formed from the immediate reward and the best Q-value in the next state.

Derivation. The update is motivated by the Bellman optimality equation for Q^*:

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right].$$

Given a sample transition (s, a, r, s'), the quantity $r + \gamma \max_{a'} Q(s', a')$ is a sampled estimate of the right-hand side. If our current estimate $Q(s, a)$ is accurate, this sample should approximately equal $Q(s, a)$. The TD error measures the discrepancy, and we use it to adjust $Q(s, a)$ toward the target.

Over many updates, this stochastic approximation procedure converges to Q^* under appropriate conditions (every state-action pair visited infinitely often, suitable learning rate schedule, bounded rewards).

Note. Q-learning is off-policy: the update uses $\max_{a'} Q(s_{t+1}, a')$, corresponding to the greedy action in the next state, regardless of which action is actually taken. This decoupling allows the agent to learn about the optimal policy while following an exploratory behavior policy (such as ϵ-greedy).

Q-learning is practical in small, discrete state-action spaces where we can maintain a table of Q-values. For each state-action pair (s, a), we store an estimate $Q(s, a)$ and update it whenever we observe a transition starting from (s, a).

15.11 Deep Q-Networks (DQN)

Tabular Q-learning becomes infeasible when the state or action spaces are large or continuous. Deep Q-Networks (DQN) address this limitation through function approximation.

15.11.1 Function Approximation

DQN approximates $Q_\theta(s, a)$ with a multilayer perceptron (MLP), where θ denotes the network parameters. The network takes a state s as input and outputs Q-values for all actions.

Training involves minimizing the squared temporal-difference error. Given a transition (s, a, r, s'), the loss is:

$$L(\theta) = \Big(r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a)\Big)^2 . \tag{43}$$

Naively minimizing this loss is unstable because the target $r + \gamma \max_{a'} Q_\theta(s', a')$ depends on the parameters θ being optimized. DQN introduces two key innovations to stabilize training:

Target Network. DQN maintains a separate target network with parameters θ^-, which is updated less frequently than the online network θ. The loss uses the target network for computing target Q-values:

$$L(\theta) = \Big(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a)\Big)^2 . \tag{44}$$

By keeping θ^- fixed for many gradient steps, the targets remain stable, reducing oscillations and divergence. Periodically, θ^- is updated to match θ.

Replay Buffer. Instead of learning from transitions in the order they occur, DQN stores transitions in a replay buffer and samples mini-batches uniformly for training.

This decorrelates consecutive transitions, which would otherwise exhibit strong temporal correlation and cause the network to overfit to recent experiences. Replay also improves data efficiency: each transition can be used for multiple gradient updates.

15.11.2 The DQN Algorithm

The complete DQN algorithm proceeds as follows:

1. Initialize replay buffer and network parameters θ, θ^-
2. For each episode:
   - Observe initial state s_0
   - For time steps $t = 0, 1, 2, \ldots$:
     (a) Select action using ϵ-greedy based on $Q_\theta(s_t, \cdot)$
     (b) Execute action, observe reward r_t and next state s_{t+1}
     (c) Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer
     (d) Sample random mini-batch from replay buffer
     (e) Compute targets using θ^- and update θ by gradient descent on squared TD error

(f) Periodically update $\theta^- \leftarrow \theta$

DQN demonstrated that deep reinforcement learning could achieve human-level performance on complex tasks such as Atari games, learning directly from raw pixel inputs. This success established neural network function approximation as a viable approach for scaling reinforcement learning to high-dimensional state spaces.

15.12 Policy-Based Methods and Policy Gradients

Value-based methods learn V^π or Q^π and then derive a policy. While value-based methods work well for discrete action spaces, they become problematic for continuous actions (where we cannot enumerate all actions to find the maximum) and for tasks requiring inherently stochastic policies. Policy-based methods instead optimize a parameterized policy $\pi_\theta(a \mid s)$ directly. This is especially attractive for continuous actions, where we can model π_θ with a Gaussian distribution whose mean and log-standard-deviation are produced by a neural network, and for discrete actions, where a softmax parameterization is typical.

15.12.1 Objective and Policy Gradient

Let $\tau = (s_0, a_0, r_0, s_1, \ldots)$ denote a trajectory and $p_\theta(\tau)$ the distribution over trajectories induced by π_θ. The standard objective is the expected discounted return:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta}\left[G_0\right], \qquad G_0 = \sum_{t=0}^{T} \gamma^t r_t. \tag{45}$$

Using the log-derivative trick, the gradient can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_0\right]. \tag{46}$$

This expression forms the basis of policy gradient algorithms.

15.12.2 REINFORCE

REINFORCE uses Monte Carlo rollouts to estimate the policy gradient. Given a batch of trajectories $\mathcal{D}$, a common surrogate loss is:

$$L^{\mathrm{PG}}(\theta) = -\frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \sum_{t=0}^{T} \log \pi_\theta(a_t^i \mid s_t^i)\, G_t^i, \tag{47}$$

where G_t is the reward-to-go:

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}. \tag{48}$$

Using G_t instead of G_0 respects causality (actions do not influence past rewards) and typically reduces variance.

Return normalization. In practice, it is common to normalize G_t (or advantages) by subtracting the batch mean and dividing by the batch standard deviation. This keeps gradient magnitudes stable across epochs and reduces sensitivity to reward scale.

Note. REINFORCE alternates between collecting rollouts with the current policy and updating θ using the sample-based loss. The resulting policy remains stochastic during training to maintain exploration, while deployment often uses a deterministic action (e.g., the mean action of a Gaussian policy).

15.13 Actor-Critic Methods

REINFORCE can suffer from high-variance gradient estimates. Actor-critic methods reduce variance by introducing a learned baseline (the critic) to compute advantages. The critic approximates the state-value function $V_\phi(s)$ and is trained via regression:

$$L^V(\phi) = \mathbb{E}\left[(V_\phi(s_t) - G_t)^2\right]. \tag{49}$$

The actor is updated with an advantage estimate rather than the raw return. The advantage $A_t = G_t - V_\phi(s_t)$ measures how much better action a_t is compared to the average action in state s_t. Using advantages as weights reduces gradient variance without introducing bias, because subtracting the state-dependent baseline $V_\phi(s_t)$ does not change the expected gradient:

$$L^{\text{A2C}}(\theta) = -\mathbb{E}\left[\log \pi_\theta(a_t \mid s_t)\, A_t\right], \qquad A_t = G_t - V_\phi(s_t). \tag{50}$$

Updating the critic multiple times per batch is common, as the critic can be optimized as a supervised learning problem.

15.13.1 Temporal-Difference Advantages

Using the full return G_t yields low bias but high variance. Temporal-difference (TD) advantages reduce variance by bootstrapping:

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \tag{51}$$

$$A_t^{\text{TD}} = \delta_t, \qquad A_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n}) - V_\phi(s_t). \tag{52}$$

These n-step estimators provide a bias-variance tradeoff between MC returns and pure TD.

15.13.2 Generalized Advantage Estimation (GAE)

GAE averages n-step estimators with an exponential weighting:

$$A_t^{\text{GAE}} = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}. \tag{53}$$

The parameter $\lambda \in [0,1]$ controls the bias-variance tradeoff; $\lambda \to 1$ approaches Monte Carlo, while $\lambda \to 0$ approaches TD.

15.14 Proximal Policy Optimization (PPO)

Actor-critic methods typically update the policy once per batch of on-policy data. PPO improves sample efficiency by reusing data with an importance ratio:

$$w_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{54}$$

To prevent overly large updates, PPO uses a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = -\mathbb{E}\left[\min\left(w_t A_t, \operatorname{clip}(w_t, 1-\epsilon, 1+\epsilon) A_t\right)\right]. \tag{55}$$

Clipping limits the improvement of good actions and the degradation of bad actions, stabilizing training while allowing multiple gradient steps on the same batch.

15.15 Entropy Regularization

To encourage exploration, many policy-gradient methods add an entropy bonus:

$$H(\pi_\theta) = \mathbb{E}_{s_t, a_t \sim \pi_\theta}\left[-\log \pi_\theta(a_t \mid s_t)\right]. \tag{56}$$

When minimizing a loss, the entropy term is typically subtracted: $L_{\text{total}} = L_{\text{policy}} + c_V L^V - c_H H(\pi_\theta)$ with $c_H > 0$. This discourages premature collapse to overly deterministic policies.

15.16 Practical Training Notes

Several implementation details strongly affect performance:

- **Stochastic vs. deterministic evaluation:** stochastic policies are used for exploration during training, while a deterministic policy (e.g., the mean action) is often evaluated at test time for repeatability.
- **Normalization:** normalizing returns or advantages stabilizes gradients and improves learning speed.
- **Data collection vs. updates:** on-policy methods must gather new data after policy updates, while PPO balances this requirement with multiple updates on recent data.

16 Active Learning

Labels are often far more expensive than unannotated data. Diagnosing rare diseases, auditing corporate fraud, or curating safety-critical driving scenarios all require expert feedback, yet only a small subset of examples truly shapes the model. Active learning asks how to collect a small but representative labeled sample under a strict budget by interactively choosing which points to query. This chapter develops three progressively richer settings:

1. A transductive problem where we explicitly distinguish between the domain we can sample from and the target region where we aim to be accurate.
2. A safety-critical optimization problem in which only safe queries are admissible.
3. A batch selection task where labeling decisions are taken in groups using a coverage heuristic.

We follow the same storyline: first define the information-theoretic principle, then demonstrate how it carries over to safe Bayesian optimization, and finally study the batch coverage formulation.

16.1 Transductive information gain

We start with the transductive view. The learner is handed a domain $\mathcal{X}$, a *target* subset $\mathcal{A} \subseteq \mathcal{X}$ on which performance is ultimately measured, and a sample space $\mathcal{S} \subseteq \mathcal{X}$ that can actually be queried. Let f be an unknown stochastic process indexed by $\mathcal{X}$. Observation noise is modeled via $y_x = f_x + \epsilon_x$, where ϵ_x is independent, mean-zero noise. At iteration n we already collected

$$\mathcal{D}_{n-1} = \{(x_i, y_i)\}_{i<n} \subseteq \mathcal{S} \times \mathbb{R},$$

and the goal is to select $x_n \in \mathcal{S}$ such that the new datum (x_n, y_{x_n}) maximally informs us about f over the target domain $\mathcal{A}$. This raises two central questions:

- How do we quantify the information gain that a particular point provides?
- Given this score, how do we efficiently pick $x_n \in \mathcal{S}$ to maximize it?

16.2 Information-based transductive learning

Information-based transductive learning (ITL) answers both questions by selecting the point that maximizes the conditional mutual information between the new observation and the restriction of f to $\mathcal{A}$:

$$x_n = \arg\max_{x \in \mathcal{S}} I\left(\{f_x\}_{x \in \mathcal{A}} ; y_x \mid \mathcal{D}_{n-1}\right)$$

When $f \sim \mathcal{GP}(\mu, k)$ is a Gaussian process with known mean and kernel, the mutual information admits a closed form,

$$I\left(\{f_x\}_{x \in \mathcal{A}} ; y_x \mid \mathcal{D}_{n-1}\right) = \frac{1}{2} \log\left(\frac{\operatorname{Var}\left(y_x \mid \mathcal{D}_{n-1}\right)}{\operatorname{Var}\left(y_x \mid \{f_x\}_{x \in \mathcal{A}}, \mathcal{D}_{n-1}\right)}\right).$$

The numerator is simply the predictive variance from the GP posterior—how uncertain we currently are about y_x before revealing its label. The denominator measures what uncertainty would remain after conditioning on the entire target restriction $\{f_x\}_{x\in\mathcal{A}}$; it shrinks when labeling x clarifies many points in $\mathcal{A}$. Therefore a large ratio pinpoints samples whose individual labels resolve substantial ambiguity precisely where performance matters. The logarithm turns this ratio into an additive gain, so selecting the point with the largest mutual information implements a principled exploration strategy that prioritizes variance reduction on $\mathcal{A}$ while respecting that we may only query within the feasible set $\mathcal{S}$.

This completes the transductive core of the chapter: we now possess an acquisition functional that explicitly connects feasible sampling locations to a target region of interest. The following sections show how the very same ingredients—separate sample and target domains, plus an information-theoretic score—power safety-critical optimization and batch selection.

16.3 Safe Bayesian optimization

We next apply ITL to a safety-critical setting: we no longer merely want to reduce uncertainty on $\mathcal{A}$, we must also ensure that every experimental design point is safe. This leads to the safe Bayesian optimization problem of maximizing an unknown stochastic process $f^\star$ over $\mathcal{X}$:

$$x^\star = \arg\max_{x \in \mathcal{X}} \mathbb{E}[f^\star(x)],$$

subject to the *safe set* $\mathcal{S}^\star = \{x \in \mathcal{X} : g^\star(x) \geq 0\}$, where $g^\star$ is another stochastic process describing safety. Iteratively we gather observations $y_i = f^\star(x_i)$ and $z_i = g^\star(x_i)$ for $x_i \in \mathcal{X}$, but future samples must not violate the unknown constraint $g^\star(x) \geq 0$. The challenge is therefore to pick x_n that improves our estimate of the maximum value while remaining in the safe region with high probability.

16.4 Safe Bayesian optimization via ITL

To enforce safety, we fit independent Gaussian processes to the objective and constraint observations. The GP posterior for $f^\star$ induces lower and upper confidence bounds $\ell_n^f(x)$ and $u_n^f(x)$ such that the posterior mean lies inside $[\ell_n^f(x), u_n^f(x)]$ with, say, 95% probability; an analogous pair $\ell_n^g(x), u_n^g(x)$ is derived for $g^\star$.

These confidence bounds define conservative and optimistic estimates of the safe region,

$$\mathcal{S}_n = \{x : \ell_n^g(x) \geq 0\}, \qquad \hat{\mathcal{S}}_n = \{x : u_n^g(x) \geq 0\},$$

and identify candidate maximizers through

$$\mathcal{A}_n = \Big\{x \in \hat{\mathcal{S}}_n : u_n^f(x) \geq \max_{x' \in \mathcal{S}_n} \ell_n^f(x')\Big\}.$$

The sample space is restricted to the provably safe set $\mathcal{S}_n$, whereas the target domain focuses on the optimistic maximizers $\mathcal{A}_n$. Applying ITL with $\mathcal{S} = \mathcal{S}_n$ guarantees that every query satisfies the safety constraint with high probability, because only points whose lower confidence bound remains non-negative are eligible. Simultaneously, setting $\mathcal{A} = \mathcal{A}_n$ targets those points most likely to improve the incumbent optimum—locations whose upper confidence bound could exceed the best certified value. Observe that the $\max_{x' \in \mathcal{S}_n} \ell_n^f(x')$ term is the best safe value known so far, so only points that might surpass it are included in $\mathcal{A}_n$.

Having adapted the transductive principle to a safe sequential design problem, we now turn to a complementary constraint: labeling must sometimes occur in *batches*. The underlying idea stays the same—identify a subset of examples whose labels will control the generalization error—but now the constraint is a fixed batch size rather than safety.

16.5 Batch active learning

Many labeling processes run in batches to leverage annotator availability. Suppose we are given an input domain $\mathcal{X}$ with distribution P over it, oracle access to an unknown function $f : \mathcal{X} \to \mathcal{Y}$, a finite population $X = \{x_1, \ldots, x_m\} \subseteq \mathcal{X}$, and a budget $b \leq m$. We must pick a subset $L \subseteq X$ of size b and request labels $\{f(x) : x \in L\}$. The goal is to choose L such that the learner trained on these labels generalizes well under P.

16.6 Auxiliary definitions

The batch formulation by Yehuda et al. (2022) relies on local homogeneity:

- For $x \in \mathcal{X}$ and $\delta > 0$, the ball $B_\delta(x) = \{x' \in \mathcal{X} : \|x - x'\| \leq \delta\}$ is *pure* if f is constant on that ball.
- For $L \subseteq \mathcal{X}$, define the covered region $C(L, \delta) = \bigcup_{x \in L} B_\delta(x)$. We write C_r for the subset on which the 1-nearest-neighbor classifier $\tilde{f}$ trained on $Z = \{(x, f(x)) : x \in L\}$ matches f, and C_w for the disagreement region.

- The impurity of radius δ is $\tilde{\pi}(\delta) = \mathbb{P}_{x\sim P}\big[B_\delta(x) \text{ is not pure}\big]$, a non-decreasing function of δ.

If a point is misclassified by the 1-NN classifier, it must either lie outside the covered region or inside a non-pure ball. Hence,

$$\mathbb{P}_{x\sim P}\big[\tilde{f}(x) \neq f(x)\big] \leq 1 - \mathbb{P}\big(C(L,\delta)\big) + \tilde{\pi}(\delta).$$

Active learning therefore seeks L and δ that minimize this upper bound. A practical strategy is to select δ first and then maximize the coverage probability $\mathbb{P}(C(L,\delta))$ under a budget constraint.

16.7 Coverage maximization and ProbCover

Given the bound above, we must solve

$$\max_{L\subseteq X,\, |L|=b} \mathbb{P}\Big(\bigcup_{x\in L} B_\delta(x)\Big),$$

but two obstacles arise: the underlying distribution P is unknown and the optimization problem is NP-hard. The fix proceeds in two steps:

1. Approximate P with the empirical distribution over X, which amounts to counting how many points lie inside the covered region.
2. Apply a greedy maximal-coverage heuristic known as ProbCover.

Let $G = (X, E)$ be a graph with edges between points at distance at most δ. ProbCover iteratively adds the point that covers the largest number of yet-uncovered neighbors:

```
$L \leftarrow \varnothing$
for $i = 1$ to $b$ do
    $x^\star \leftarrow \arg\max_{x\in X} |\{x' : (x,x') \in E,\ x' \text{ not yet covered}\}|$
    $L \leftarrow L \cup \{x^\star\}$
    Remove all edges incident to points in $B_\delta(x^\star) \cap X$ (they need not be considered again)
end for
return $L$
```

This greedy routine provides a logarithmic approximation guarantee for set coverage problems and serves as a practical batch active learning policy: small δ focuses on fine-grained distinctions (lower impurity, smaller coverage), whereas large δ broadens coverage but risks mixing labels. By tuning δ and b, we trade off robustness of the 1-NN classifier against the number of labels queried.

16.8 Summary

We began by splitting the world into a target region and a feasible sampling set, quantifying information gain through ITL. We then reused this perspective to design a safe Bayesian optimization loop where feasibility is enforced through GP confidence bounds, and finally transitioned to batch selection by casting informativeness as probabilistic coverage.

17 Counterfactual Invariance

Machine-learning estimators often achieve high accuracy for the wrong reasons. Spurious correlations, confounders, and unobserved interventions can create shortcuts that models exploit during training but that fail under distribution shift. Counterfactual invariance addresses this failure mode by demanding that a predictor remain stable when nuisance factors are changed while the underlying causal mechanism of interest is held fixed. This chapter develops the idea systematically: starting from motivating pathologies, we introduce the causal-graph machinery necessary to reason about counterfactual statements, derive formal conditions for counterfactual invariance in both causal and anti-causal regimes, and finally discuss how to enforce those conditions in practice.

17.1 Motivation: when correlations lie

Classical statistical anecdotes already hint at the perils of naive correlation chasing:

- Observational studies once concluded that young children sleeping with a light on were more likely to develop myopia; later it was shown that myopic parents both left lights on and passed on their genetics.
- Correlations between mozzarella consumption and doctorates awarded, or between lice and health, are artifacts of shared confounders rather than direct causation.
- In medical imaging, Badgeley et al. found that a hip-fracture classifier had learned to use hospital-specific devices visible in X-rays rather than skeletal cues, leading to dramatic failures under domain shift.

These examples illustrate three recurring pitfalls: *reverse causation* (target causes features), *third-cause fallacies* (hidden confounders drive both feature and target), and *shortcut learning* (models latch onto non-causal proxies). Counterfactual invariance seeks representations and predictors that respond only to the causal content of the data.

17.2 Shortcut learning and domain shifts

Shortcut learning arises when causal and spurious factors co-occur in the training data. Given two environments $\mathcal{E}_1$ and $\mathcal{E}_2$ whose feature distributions differ via nuisance features W, an estimator f trained on $\mathcal{E}_1$ may exploit W as a proxy for the label Y rather than responding to the causal features X. At test time, a domain shift—either because W changes distribution or because we deploy f in a new environment—breaks this proxy, and accuracy collapses. The solution is to encode *invariant representations*: features of X that do not depend on the environment W yet retain the signal about Y.

17.3 Counterfactuals and invariance

Let X be a random feature vector representing an object, and let Y be the target variable we wish to predict. Suppose we also observe a random vector W capturing nuisance attributes: factors that may influence X but should not influence the prediction rule. A counterfactual $X(w)$ is the feature vector we would observe if we were to intervene and set W to the specific value w, leaving everything else unchanged. Formally, the intervention replaces the generative mechanism for W with the constant w while keeping the structural equations for other variables intact.

Definition 5 (Counterfactual invariance) *A predictor f is* counterfactually invariant *with respect to W if for any two values w, w' in the range of W and every realization of X we have*

$$f\big(X(w)\big) = f\big(X(w')\big).$$

In words, if we edit the nuisance factors while holding everything else constant, the prediction remains unchanged.

Intuitively, counterfactual invariance demands that f ignore all pathways through which W can influence the prediction except via the causal content shared across interventions.

17.4 Two causal regimes

The causal structure of the problem determines how we should enforce invariance. Two canonical regimes cover most applications:

- **Causal regime.** Changing features X has a causal effect on Y, e.g., in a medical-diagnosis setting where symptoms influence the disease classification. Here we typically assume $X \to Y$.
- **Anti-causal regime.** The target Y causes the features X, e.g., in disease detection where the disease (cause) produces observable symptoms (effects). Here we model $Y \to X$.

In both regimes the nuisance W may influence X but must not affect Y. The difference lies in whether Y lies downstream or upstream of X in the causal graph, which alters the conditional independences we can exploit.

17.5 Causal graphs and d-separation

To reason formally, we represent the relationships among W, X, Y, and latent variables through a causal graph. Nodes correspond to random variables, directed edges denote causal influence, and we assume the graph is a directed acyclic graph (DAG). For example, in the causal regime (where $X \to Y$) a minimal graph might include:

$$W \to X \to Y,$$

with potential latent confounders U influencing both W and X, or selection variables S governing which samples appear in the dataset. In the anti-causal regime, the edge direction flips: $Y \to X$, yet W still influences X.

Conditional independences implied by the graph are characterized via *d-separation.* A path between two nodes is blocked if it contains:

- A chain $A \to B \to C$, $A \leftarrow B \to C$, or $A \leftarrow B \leftarrow C$ where the middle node B is conditioned on.
- A collider $A \to B \leftarrow C$ where the middle node B is *not* conditioned on (nor any of its descendants).

If every path between two nodes is blocked given a conditioning set Z, the nodes are d-separated, implying conditional independence. This rule lets us derive necessary independences for counterfactual invariance.

Intuition. D-separation tracks whether there is still an “open pipeline” along which information can travel. In a chain or fork, every influence flowing from one endpoint to the other must pass through the middle node, so once we observe that node the pipeline is saturated—learning B already captures everything A could reveal about C. A practical fork example is a genetic mutation B that simultaneously raises cholesterol A and blood pressure C: before measuring the gene, high cholesterol hints at high blood pressure; after conditioning on the mutation, the endpoints decouple because their only link has been cut. Likewise, in the chain $C \to B \to A$ (smoking $\to$ tar buildup $\to$ chronic cough) conditioning on tar levels breaks the association between smoking and cough, because tar already summarizes whatever smoking would have conveyed. Colliders behave oppositely: two independent causes $A \to B \leftarrow C$ remain disconnected unless we observe the collision (e.g., conditioning on sneezing that can be caused by both a cold and allergies), in which case the path becomes active and learning one cause “explains away” the other.

17.6 Confounding and selection pitfalls

Two additional phenomena can break counterfactual invariance if unaddressed:

- **Confounding.** Hidden variables U that influence both W and X (or Y) create associations that do not disappear under interventions. Simpson’s paradox—where aggregated data reverses the trend observed within subgroups—is a textbook manifestation.

- **Selection bias.** Conditioning on a selection variable S that depends on W and X introduces collider bias: even independent variables become correlated when we restrict attention to samples with $S = 1$ (e.g., only applicants on LinkedIn).

Accounting for these effects requires either modeling the hidden variables explicitly or designing algorithms that enforce the necessary conditional independences despite selection.

17.7 Necessary conditions for invariance

Counterfactual invariance imposes specific conditional independence relations on $f(X)$. Intuitively, if f ignores W even under interventions, then after conditioning on the causal parents, the prediction cannot leak information about W. The following conditions are necessary:

- **Anti-causal scenario:** $f(X) \perp W \mid Y$. Once we know the true label Y, the prediction $f(X)$ must reveal nothing about the nuisance W. Otherwise, altering W while holding Y fixed would change f.
- **Causal scenario without selection:** $f(X) \perp W$. Because Y is downstream of X, any dependence on W indicates that f has retained nuisance information.
- **Causal scenario with selection bias:** If selection variables S depend on both W and X, then colliders induced by conditioning on $S = 1$ can reintroduce spurious correlations. In this case, a conservative requirement is $f(X) \perp W \mid Y$, provided that Y does not depend on the selection mechanism once X and W are fixed.

Below we sketch why these conditions follow from d-separation.

Anti-causal proof sketch. Assume $Y \to X$ and $W \to X$, possibly with confounders capturing unobserved causes of W and Y. Counterfactual invariance means $f(X)$ only depends on the portion of X that remains invariant under interventions on W, typically written as $X_W^\perp$. In the anti-causal graph, every path from $X_W^\perp$ to W is blocked once we condition on Y: colliders opened by Y are observed, while colliders through latent variables remain unobserved and thus block the path. Therefore, if f only uses $X_W^\perp$, we must have $f(X) \perp W \mid Y$.

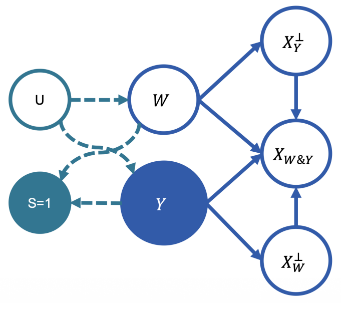


Figure 4: Anti-causal regime

Causal proof sketch. When $X \to Y$, f can depend on X directly. Yet to be invariant, f must ignore all components of X that are influenced by W. In the absence of selection, the graph ensures that every path from $X_W^\perp$ to W goes through a collider—either $X_{W\&Y}$ or Y itself—that is not observed. Consequently, the residual representation cannot correlate with W, leading to the unconditional independence $f(X) \perp W$. If we condition on a selection variable, some paths may become unblocked, so we revert to the more cautious requirement conditioned on Y.

17.8 Enforcing invariance via distribution matching

The independence conditions can be operationalized by matching the distributions of the predictions across environments. Consider the anti-causal setting with binary W and Y. Counterfactual

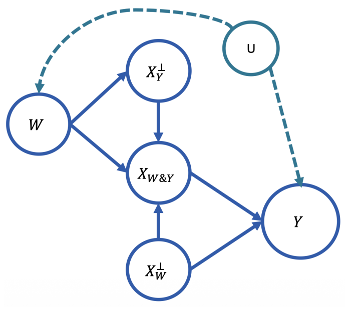


Figure 5: Causal regime

invariance demands

$$f(X) \mid \{W = w,\, Y = y\} \quad \overset{d}{=} \quad f(X) \mid \{W = w',\, Y = y\}$$

for all w, w', y. In practice we can:

1. Choose a discrepancy measure $\Delta(p, q)$ between probability distributions (e.g., maximum mean discrepancy, Wasserstein distance, or KL divergence).
2. Add a regularization term that penalizes discrepancies between the empirical distributions of $f(X)$ across the nuisance groups conditioned on Y:

$$\mathcal{L}_{\text{inv}} = \sum_{y} \sum_{w,w'} \Delta\Big(\hat{p}\big(f(X) \mid W = w, Y = y\big),\ \hat{p}\big(f(X) \mid W = w', Y = y\big)\Big).$$

3. Optimize the predictor to minimize the original task loss plus $\lambda \mathcal{L}_{\text{inv}}$, where λ balances task fit and invariance.

This strategy enforces that, within each class y, the representation used by f carries no information about the nuisance. Extensions include adversarial training where a discriminator tries to predict W from the learned representation while the encoder attempts to fool it, or explicit data augmentation with synthetically generated counterfactuals when such interventions are available.

17.9 Summary

Counterfactual invariance provides a principled remedy for shortcut learning. By explicitly modeling the nuisance factors W, constructing counterfactuals $X(w)$, and reading off the necessary conditional independences from causal graphs, we can reason about when a predictor truly focuses on causal features. Enforcing these independences—through distribution matching, adversarial debiasing, or counterfactual augmentation—yields models whose predictions remain stable even when environments shift. In domains where decisions must rest on the right reasons, such invariance is not merely desirable; it is essential.

18 Variational Autoencoders

Learning useful representations without labels requires us to tell the model what "useful" means. Variational autoencoders (VAEs) formalize this request by combining the information-theoretic appeal of autoencoders with explicit probabilistic modeling. This chapter develops VAEs from the bottom up: we motivate the desiderata for unsupervised representations, examine why maximizing mutual information alone fails, cast autoencoders in a Bayesian light, derive the evidence lower bound (ELBO), and describe how to optimize it in practice.

18.1 Representation learning without supervision

Deep networks can be viewed as compositions of an *encoder* that maps the input $x \in \mathcal{X}$ into a latent representation z and a *decoder* (or predictor) that turns z into a task output. When no labels are available, the only supervision we can provide is the input itself. Good representations should therefore satisfy three properties:

Informative. The original input should be recoverable from z, so no essential information is discarded.

Disentangled. Individual coordinates of z should align with distinct generative factors (pose vs. lighting, stroke width vs. digit identity, *etc.*), enabling controlled manipulation.

Robust. Small perturbations in the input should not cause drastic changes in z, and conversely modifying z slightly should not flip the reconstruction arbitrarily.

Autoencoders optimize informativeness by minimizing reconstruction error, but disentanglement and robustness typically require further regularization.

18.2 The infomax principle and its limitations

Let $\text{enc}_\theta \in \mathcal{H}$ be a (possibly stochastic) encoder. The infomax principle (Linsker, 1988) advocates selecting the parameters θ that maximize the mutual information $I(X; Z)$ between inputs X and their representations $Z = \text{enc}_\theta(X)$:

$$I(X; Z) = \int p(x, z) \log \frac{p(x, z)}{p(x)\, p(z)}\, dx\, dz$$

With only data $x_1, \ldots, x_n$, the expectation can be approximated via Monte Carlo:

$$I(X; Z) \approx \sum_{i \le n} \mathbb{E}_{Z|x_i} \left[\log p\left(x_i \mid Z\right)\right]$$

which resembles the reconstruction score of an autoencoder. Unfortunately, maximizing $I(X; Z)$ alone can be trivial whenever $\mathcal{X}$ and $\mathcal{Z}$ are rich enough: the encoder can simply implement a bijection (or memorize the dataset) so that Z is a lossless copy of X. Such degenerate solutions are perfectly informative but neither disentangled nor robust, underscoring the need for additional inductive biases.

18.3 Latent-variable generative modeling

To go beyond empty informativeness, we posit a generative story for the data. A latent variable Z is sampled from a prior $p(z)$, and the observation X is drawn from a conditional decoder $p_\theta(x \mid z)$. This *decoder* can be a deep neural network that outputs the parameters of a likelihood distribution (Gaussian for continuous data, Bernoulli/Categorical for binary or discrete data). High-quality representations now correspond to posterior inferences: given x, infer the distribution $p_\theta(z \mid x)$ of latent causes. Integrating the latent variable out gives the marginal likelihood $p_\theta(x) = \int p_\theta(x \mid z)p(z)\, dz$, which quantifies how well the generative model explains the observation.

Two conceptual benefits arise:

- Sampling new x is straightforward: draw $z \sim p(z)$ and decode it. VAEs thus act as generative models capable of "hallucinating" plausible data such as handwritten digits or faces.

- Regularization becomes Bayesian: the prior $p(z)$ codifies which representations are plausible (e.g., standard normal means centered, isotropic latent factors), discouraging memorization.

The main obstacle is posterior inference. Computing $p_\theta(z \mid x)$ exactly is typically intractable because it requires normalizing the product $p_\theta(x \mid z)p(z)$ over all z. Variational inference circumvents this by introducing an auxiliary encoder $q_\phi(z \mid x)$, parameterized by ϕ, that approximates the true posterior.

18.4 A manifold perspective

High-dimensional data often concentrate near a *manifold*—a smooth, low-dimensional surface embedded in the ambient space. Pictures of faces, for instance, span thousands of pixels yet vary along a handful of semantic axes such as pose, lighting, and expression. VAEs can be interpreted as tools for learning coordinates on this manifold. The decoder $p_\theta(x \mid z)$ acts like a chart that maps latent coordinates z into points on (or near) the data manifold, while the encoder $q_\phi(z \mid x)$ performs the inverse mapping from observations back to manifold coordinates. The prior $p(z)$ then regularizes the geometry of this manifold by favoring latents near the origin, which in turn encourages smooth, disentangled directions. Thinking in manifold terms clarifies why VAEs produce interpolations that remain realistic: straight lines in latent space correspond to geodesic-like curves on the learned manifold, yielding gradual transitions in pixel space.

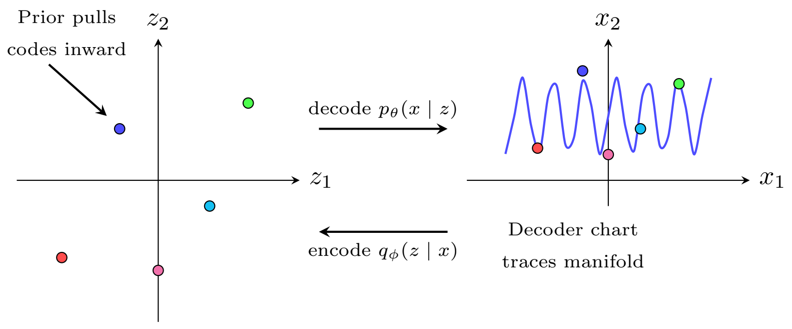

Prior pulls codes inward
z_2
z_1
decode $p_\theta(x \mid z)$
encode $q_\phi(z \mid x)$
x_2
x_1
Decoder chart traces manifold

Figure 6: The encoder learns manifold coordinates while the decoder maps latent points back onto the curved data manifold, keeping interpolations smooth.

18.5 A Bayesian detour

Bayesian estimation provides intuition for why priors help disentangle noise from signal. Suppose we wish to estimate the mean shoe size μ in a small town. A Gaussian prior $\mu \sim \mathcal{N}(\alpha, \beta)$ captures our belief before collecting data. Observing measurements $x_1, \ldots, x_n$ with $x_i \sim \mathcal{N}(\mu, 1)$ yields a Gaussian posterior $\mathcal{N}(\mu \mid m, s^2)$ that balances the prior and the evidence. When few samples are available, the posterior stays near α (robustness to noise); as n grows, the data dominate (informativeness). VAEs apply the same logic in every latent dimension: the prior $p(z)$ nudges encodings toward structured, disentangled regions while still allowing the decoder to reconstruct the input accurately.

19 Non-parametric Bayesian Methods

How can we endow probabilistic models with enough flexibility to grow with the data instead of committing to a fixed number of parameters? Non-parametric Bayesian methods answer this question by replacing finite-dimensional priors with distributions over infinite-dimensional objects such as probability measures. This chapter traces the path from Bayesian inference for a single Gaussian to Gaussian mixture models (GMMs) with an unbounded number of clusters, highlighting the role of conjugate priors, Gibbs sampling, Dirichlet processes, and exchangeability.

19.1 Warm-up: Bayesian inference for a single Gaussian

Consider one-dimensional observations $X = \{x_1, \dots, x_n\}$ drawn i.i.d. from $\mathcal{N}(\mu, \sigma^2)$ with known variance $\sigma^2 = 1$ but unknown mean μ. Bayesians update a prior belief $\mu \sim \mathcal{N}(m_0, k_0^{-1})$ via Bayes' rule:

$$p(\mu \mid X) \propto p(X \mid \mu)\, p(\mu) = \left(\prod_{i=1}^{n} \mathcal{N}(x_i \mid \mu, 1)\right) \mathcal{N}(\mu \mid m_0, k_0^{-1}).$$

Because Gaussians are conjugate to themselves, the posterior remains Gaussian with parameters

$$m_n = \frac{k_0 m_0 + n\bar{x}}{k_0 + n}, \qquad k_n = k_0 + n,$$

where $\bar{x}$ is the sample mean. Intuitively, k_0 quantifies how confident we were in m_0; after seeing n data points, the precision simply adds up. The posterior mean m_n is a weighted average between the prior mean and the data mean, showcasing regularization against outliers and a principled notion of uncertainty.

19.2 Multivariate Gaussians and conjugate priors

For d-dimensional data $x_i \sim \mathcal{N}(\mu, \Sigma)$ with both μ and Σ unknown, the normal-inverse-Wishart (NIW) distribution is a conjugate prior whose hyperparameters carry clear meaning:

- $m_0 \in \mathbb{R}^d$: prior mean vector,
- $k_0 > 0$: scaling factor controlling confidence in m_0,
- $S_0 \in \mathbb{R}^{d \times d}$: scale matrix encoding prior beliefs about covariance structure,
- $\nu_0 > d - 1$: degrees of freedom, controlling confidence in S_0.

Crucially, the inverse-Wishart component enforces that sampled covariance matrices remain positive semi-definite, which is required for any valid multivariate Gaussian. The NIW density is

$$p(\mu, \Sigma) = \mathrm{NIW}(\mu, \Sigma \mid m_0, k_0, S_0, \nu_0).$$

Given a dataset X, Bayes' rule yields the posterior

$$p(\mu, \Sigma \mid X) = \mathrm{NIW}(m_n, k_n, S_n, \nu_n),$$

with updates

$$\begin{aligned} k_n &= k_0 + n, & \nu_n &= \nu_0 + n, \\ m_n &= \frac{k_0 m_0 + n\bar{x}}{k_0 + n}, & S_n &= S_0 + S_X + \frac{k_0 n}{k_0 + n}(\bar{x} - m_0)(\bar{x} - m_0)^\top, \end{aligned}$$

where S_X is the sample covariance matrix. Only sufficient statistics (mean and covariance) are required, making posterior updates computationally light even for large n.

19.3 Sampling with semi-conjugate priors

Fully conjugate priors are convenient but sometimes too rigid—for instance, we may want to express separate beliefs about the location and spread of the data. Under an NIW prior the strength of the prior mean and the tightness of the covariance are linked through k_0, so tightening one automatically tightens the other. Semi-conjugate priors break this coupling while keeping *conditionally* conjugate updates, which is all Gibbs sampling needs. A typical choice specifies

$$\mu \sim \mathcal{N}(m_0, V_0), \qquad \Sigma \sim \mathrm{IW}(S_0, \nu_0),$$

so the joint density no longer has a closed form but the conditional posteriors do. The Given current samples, Gibbs sampling iterates:

$$\mu \mid \Sigma, X \sim \mathcal{N}\left(m_p, V_p\right)$$

$$\Sigma \mid \mu, X \sim \mathrm{IW}\left(S_p, v_p\right)$$

Each step has the same flavor as the fully conjugate update: the data provide sufficient statistics, while the prior contributes virtual observations. Even though the joint posterior $p(\mu, \Sigma \mid X)$ cannot be written explicitly, the Markov chain that alternates these two conditional draws converges to it under mild conditions. Practical samplers further exploit graphical- model independencies (via d-separation) and Rao-Blackwellization to reduce variance and shorten burn-in.

Gibbs sampler details. A single Gibbs sweep for this semi-conjugate model proceeds as follows:

1. Given the current covariance sample $\Sigma^{(t)}$, draw a new mean by sampling $\mu^{(t+1)}$ from the Gaussian conditional above (using $\Sigma^{(t)}$ inside the formula).
2. Plug $\mu^{(t+1)}$ into the inverse-Wishart conditional to draw the next covariance sample $\Sigma^{(t+1)}$.
3. Repeat these two steps for many iterations, discarding the first T_{burn} draws as burn-in and optionally thinning the remainder.

Because each conditional depends on the latest value of the other block, the chain "zig-zags" through the (μ, Σ) space but still converges to the true joint posterior. Conditional independence structure (e.g., between clusters in a mixture) allows updating blocks in parallel or integrating out nuisance variables before running the sampler.

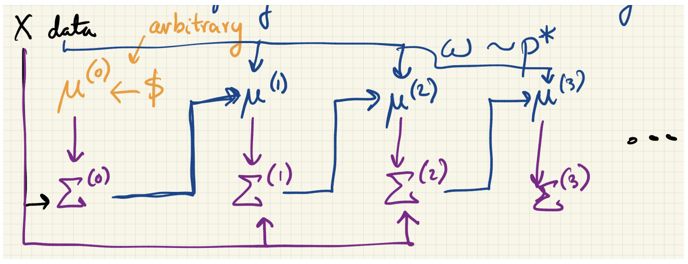


Figure 7: Gibbs sampling illustration.

19.4 Gibbs sampling in a nutshell

Whenever the posterior $p(\Theta \mid X)$ factorizes into conditionals that are easy to sample from, *Gibbs sampling* provides a route to approximate inference even if the joint density is intractable. Let $\Theta = (\Theta_1, \ldots, \Theta_\ell)$ be the latent variables of interest. Gibbs sampling constructs a Markov chain $\{\Theta^{(t)}\}_{t=0}^{\infty}$ by iterating:

1. Initialize $\Theta^{(0)}$ arbitrarily (e.g., random cluster assignments).
2. For $t = 0, 1, 2, \ldots$:
   (a) Sample $\Theta_1^{(t+1)} \sim p(\Theta_1 \mid \Theta_2^{(t)}, \ldots, \Theta_\ell^{(t)}, X)$.

(b) Sample $\Theta_2^{(t+1)} \sim p(\Theta_2 \mid \Theta_1^{(t+1)}, \Theta_3^{(t)}, \ldots, \Theta_\ell^{(t)}, X)$.

(c) Continue cycling through all coordinates until $\Theta_\ell^{(t+1)}$ is sampled given the most recent values of the others.

Each conditional draw is typically conjugate (or otherwise tractable) because all variables except one are held fixed. Under mild regularity conditions this Markov chain has $p(\Theta \mid X)$ as its stationary distribution, so samples collected after a burn-in period approximate the true posterior. In practice we discard the first T_{burn} iterations, then thin or average the remaining ones to estimate expectations. The method shines in models such as GMMs where conditionals like $p(z_i \mid z_{-i}, \pi, \mu, \Sigma, X)$ or $p(\mu_k, \Sigma_k \mid z, X)$ admit closed forms.

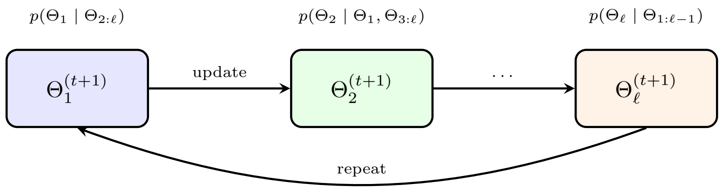

$p(\Theta_1 \mid \Theta_{2:\ell})$
$p(\Theta_2 \mid \Theta_1, \Theta_{3:\ell})$
$p(\Theta_\ell \mid \Theta_{1:\ell-1})$
$\Theta_1^{(t+1)}$
update
$\Theta_2^{(t+1)}$
$\cdots$
$\Theta_\ell^{(t+1)}$
repeat

Figure 8: Gibbs Sampling

20 Probably Approximately Correct Learning

Probably Approximately Correct (PAC) learning formalizes the intuition that a model should capture the regularities in the data-generating process, not just the accidents of a finite dataset. The framework asks how much data a learning algorithm needs before the hypothesis it outputs generalizes, and it does so without assuming a particular distribution for the inputs.

20.1 From Empirical Patterns to Guarantees

Every supervised learning pipeline starts from a labeled sample $Z = \{(x_i, y_i)\}_{i=1}^n$ drawn from an unknown distribution. Fitting a hypothesis that performs well on Z is easy; guaranteeing that the same hypothesis will predict unseen examples demands more structure. Statistical learning theory introduces the *generalization error*

$$R(\hat{c}) = \mathbb{P}_{X\sim D}\big(\hat{c}(X) \neq c(X)\big),$$

which measures how often hypothesis $\hat{c}$ disagrees with the (unknown) target concept c under the true distribution D on the instance space $\mathcal{X}$. Because D and c are inaccessible, the learner minimizes the empirical error

$$\widehat{R}_n(\hat{c}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{\hat{c}(x_i) \neq y_i\},$$

hoping that $\widehat{R}_n$ is a faithful estimate of R. PAC learning quantifies when this hope is justified.

20.2 Instance, Concept, and Hypothesis Spaces

An *instance space* $\mathcal{X}$ enumerates all objects the learner may observe. A *concept* c is a subset of $\mathcal{X}$ or, equivalently, a label function $c : \mathcal{X} \to \{0, 1\}$. A *concept class* $\mathcal{C}$ collects candidate targets, whereas a *hypothesis class* $\mathcal{H}$ contains the functions the algorithm is allowed to output. The learner receives Z with labels $y_i = c(x_i)$ (realizable setting) or draws from a distribution on $\mathcal{X} \times \{0, 1\}$ (agnostic setting) and must pick $\hat{c} \in \mathcal{H}$ that approximates c.

20.3 The PAC Criterion

A learning algorithm A is a PAC learner for concept class $\mathcal{C}$ if there exists a polynomial p such that for any distribution D on $\mathcal{X}$, any tolerance parameters $0 < \varepsilon, \delta < 1/2$, and any target $c \in \mathcal{C}$, the algorithm produces $\hat{c} \in \mathcal{H}$ satisfying

$$\mathbb{P}_{Z\sim D^n}\left(R(\hat{c}) \leq \varepsilon\right) \geq 1 - \delta$$

whenever it receives at least $n \geq p(1/\varepsilon, 1/\delta, \mathrm{size}(c))$ samples. The polynomial captures *sample complexity*; ε controls accuracy, and δ controls confidence. Efficient PAC learners also run in time polynomial in the same quantities.

20.4 Axis-Aligned Rectangles as a Running Example

To build intuition, consider $\mathcal{X} = \mathbb{R}^2$ and let $\mathcal{C}$ be all axis-aligned rectangles. Given positive and negative labeled points, a natural learner outputs $\hat{R}$, the smallest rectangle containing every positive sample.

Intuitively, this rectangle expands just enough to explain the data, so it should not misclassify too many points outside the observed cloud. The theory turns this intuition into a guarantee.

Partition the true rectangle R into four thin *strips*: upper, lower, left, and right, each with probability mass $\varepsilon/4$ under D. Let the event $\hat{R}_{\mathrm{IG}}$ denote that the learned rectangle intersects all strips ("IG" for *is good*). If a strip is missed, that entire portion of R will be falsely labeled negative, contributing at least $\varepsilon/4$ error. Conversely, intersecting every strip ensures that the area where $\hat{R}$ differs from R has probability at most ε.

The key insight is that if the learned rectangle $\hat{R}$ misses any strip, it will incur large error. We formalize this by analyzing the probability that $\hat{R}$ intersects all four strips.

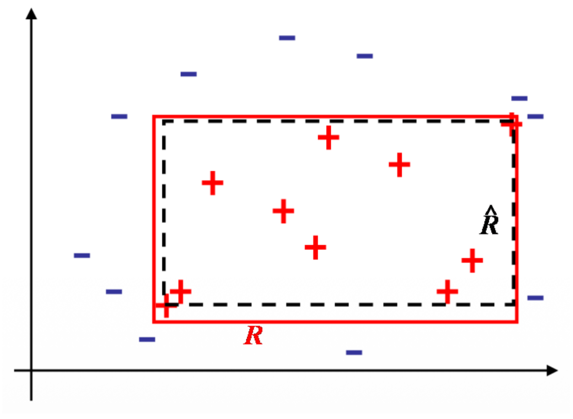


Figure 9: Axis-aligned rectangle learning: the smallest rectangle $\hat{R}$ containing all positive samples.

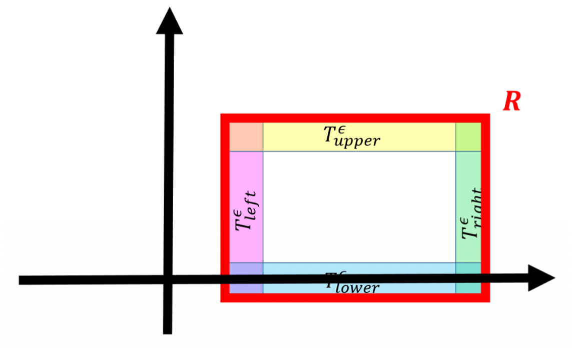


Figure 10: Partitioning the true rectangle into four strips, each with probability mass $\varepsilon/4$.

Derivation. The chance a fixed strip receives no sample is $(1-\varepsilon/4)^n \le \exp(-n\varepsilon/4)$ by $1+x \le e^x$. A union bound over the four strips yields

$$\mathbb{P}\left(\hat{R}_{\mathrm{IG}}\right) \ge 1 - 4\exp\left(-\frac{n\varepsilon}{4}\right).$$

On $\hat{R}_{\mathrm{IG}}$, the symmetric difference $R \triangle \hat{R}$ has probability at most ε, so

$$\mathbb{P}\big(R(\hat{R}) \le \varepsilon\big) \ge \mathbb{P}\left(\hat{R}_{\mathrm{IG}}\right).$$

Intuitively, intersecting every strip forces $\hat{R}$ to stretch all the way to each face of the true rectangle: if it were to stop short (say, at the top), it would exclude the sample that witnessed that strip, contradicting minimality. Consequently the only region where R and $\hat{R}$ can disagree is confined to the four strips themselves, whose total probability mass is at most ε. Bounding the disagreement set is therefore equivalent to bounding $R(\hat{R})$.

It suffices to set $n \ge \frac{4}{\varepsilon}\log\frac{4}{\delta}$ to make the failure probability at most δ. The dependence is logarithmic in $1/\delta$ and linear in $1/\varepsilon$, exactly the scaling promised by the PAC definition.

20.5 Induction Principles and Empirical Risk Minimization

Learning can be viewed as an induction principle: from observed labeled samples we induce a rule that will be used to classify new points. Once a hypothesis has been induced, deduction applies the rule to unseen inputs, while *transduction* skips explicit model building and predicts labels for a specific test set directly. PAC learning focuses on induction and the conditions under which it is justified.

The standard induction rule is *empirical risk minimization* (ERM). Given a hypothesis class $\mathcal{C}$,

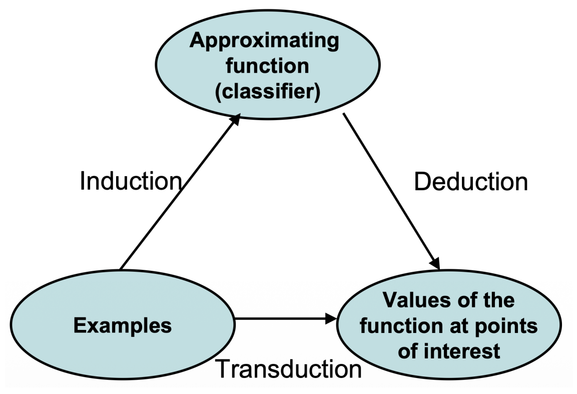

Approximating function (classifier)
Induction
Deduction
Examples
Values of the function at points of interest
Transduction

Figure 11: Relation between induction, deduction, and transduction.

ERM selects

$$\hat{c}_n^* \in \arg\min_{c\in\mathcal{C}} \widehat{R}_n(c),$$

where $\widehat{R}_n(c)$ is the empirical classification error. This choice is computable without any prior assumptions on D, but it shifts the burden to analysis: we need distribution-independent bounds on the excess risk

$$R(\hat{c}_n^*) - \inf_{c\in\mathcal{C}} R(c).$$

The rest of the chapter develops tools to control this deviation with high probability.

20.6 Uniform Convergence and the VC Inequality

The empirical minimizer is data-dependent, so the law of large numbers for a *fixed* classifier does not directly apply. For any fixed c, the law would guarantee $\widehat{R}_n(c) \to R(c)$ as $n \to \infty$, but ERM selects $\hat{c}_n$ after seeing the data, so the hypothesis itself changes with the sample. The remedy is *uniform convergence*: ensure that *all* hypotheses in $\mathcal{C}$ have empirical risks close to their true risks simultaneously. This solves the issue because it guarantees that whichever $\hat{c}_n$ ERM picks will still have $\widehat{R}_n(\hat{c}_n) \approx R(\hat{c}_n)$.

Let $c^\star = \arg\min_{c\in\mathcal{C}} R(c)$ denote the best-in-class classifier. Then

$$\begin{aligned}
\mathcal{R}\left(\hat{c}_n^\star\right) - \inf_{c\in\mathcal{C}} \mathcal{R}(c) &= \mathcal{R}\left(\hat{c}_n^\star\right) - \hat{\mathcal{R}}_n\left(\hat{c}_n^\star\right) + \hat{\mathcal{R}}_n\left(\hat{c}_n^\star\right) - \inf_{c\in\mathcal{C}} \mathcal{R}(c) \\
&\le \underbrace{\mathcal{R}\left(\hat{c}_n^\star\right) - \hat{\mathcal{R}}_n\left(\hat{c}_n^\star\right)} + \underbrace{\hat{\mathcal{R}}_n\left(c^\star\right) - \mathcal{R}\left(c^\star\right)} \\
&\le \sup_{c\in\mathcal{C}} \left|\hat{\mathcal{R}}_n(c) - \mathcal{R}(c)\right| + \sup_{c\in\mathcal{C}} \left|\hat{\mathcal{R}}_n(c) - \mathcal{R}(c)\right| \\
&\le 2\sup_{c} \left|\hat{\mathcal{R}}_n(c) - \mathcal{R}(c)\right|
\end{aligned}$$

Consequently,

$$\mathbb{P}\left(R(\hat{c}_n^*) - \inf_{c\in\mathcal{C}} R(c) > \varepsilon\right) \le \mathbb{P}\left(\sup_{c\in\mathcal{C}}\left|\widehat{R}_n(c) - R(c)\right| > \varepsilon/2\right).$$

Interpretation: the only way ERM can be more than ε worse than the best-in-class classifier is if *some* hypothesis in the class has its empirical risk misestimate the true risk by more than $\varepsilon/2$. Thus controlling uniform deviation is sufficient to control excess risk. This bound explains why empirical minimizers can have smaller *empirical* error than the true minimizer $c^\star$ yet still generalize: what matters is the uniform deviation between empirical and true risks across the class.

20.7 Finite Hypothesis Classes and Consistency

For a finite hypothesis class $\mathcal{H}$ with $N = |\mathcal{H}|$, concentration inequalities make uniform convergence explicit. For any fixed h, the empirical risk is an average of Bernoulli errors in $[0,1]$, so Hoeffding's inequality gives

$$\mathbb{P}\left(\left|\widehat{R}_n(h) - R(h)\right| > \varepsilon\right) \le 2\exp(-2n\varepsilon^2).$$

Applying a union bound over N hypotheses yields

$$\mathbb{P}\left(\sup_{h\in\mathcal{H}}\left|\widehat{R}_n(h) - R(h)\right| > \varepsilon\right) \le 2N\exp(-2n\varepsilon^2).$$

Equivalently, with probability at least $1-\delta$, the following *uniform confidence interval* holds for all $h \in \mathcal{H}$:

$$R(h) \le \widehat{R}_n(h) + \sqrt{\frac{\log N + \log(2/\delta)}{2n}}.$$

The variance term decays as $1/\sqrt{n}$ and grows only logarithmically with N, which makes large but finite hypothesis classes viable.

In the realizable case, where the algorithm can return a hypothesis consistent with the sample ($\widehat{R}_n(\hat{c}) = 0$), a sharper argument is possible. Every $h \in \mathcal{H}$ with true error $R(h) > \varepsilon$ must mislabel at least an ε-fraction of the instance space, so the probability that none of the n samples expose its error region is at most $\exp(-n\varepsilon)$. A union bound over all N bad hypotheses yields

$$\mathbb{P}\big(R(\hat{c}) > \varepsilon\big) \le |\mathcal{H}|\exp(-n\varepsilon).$$

Equivalently, for any $\delta > 0$, it suffices to take

$$n \ge \frac{1}{\varepsilon}\left(\log|\mathcal{H}| + \log\frac{1}{\delta}\right)$$

to guarantee $R(\hat{c}) \le \varepsilon$ with probability at least $1-\delta$. The logarithmic dependence on $|\mathcal{H}|$ demonstrates that even exponentially large hypothesis spaces can be learnable, provided the algorithm searches them effectively.

20.8 Agnostic PAC Learning and Bayes Risk

The realizable assumption breaks when identical feature vectors carry different labels, as in noisy measurement regimes. In this *agnostic* scenario the benchmark is the Bayes optimal classifier $c_{\text{Bayes}}(x) = \arg\max_{y\in\{0,1\}} \mathbb{P}(y \mid x)$ with minimal achievable risk $R^\star$. Unlike the realizable case, no algorithm can guarantee error below $R^\star$ if the true labeling rule lies outside the hypothesis class. Instead we ask the learner to track the best rule available inside $\mathcal{C}$.

A hypothesis class $\mathcal{C}$ is agnostically PAC learnable if there exists a learning algorithm A that, for every distribution on $\mathcal{X} \times \{0,1\}$ and every ε, δ, outputs $\hat{c}$ with

$$\mathbb{P}\left(R(\hat{c}) - \inf_{c\in\mathcal{C}} R(c) \le \varepsilon\right) \ge 1-\delta.$$

The learner now competes with the best function inside $\mathcal{C}$ rather than with the Bayes classifier itself. This shift has two consequences: (i) the comparison point is $R^\star_{\mathcal{C}} := \inf_{c\in\mathcal{C}} R(c)$, the approximation error induced by the hypothesis class, and (ii) the sample complexity typically increases because the learner must control both estimation error (difference between $\hat{c}$ and the empirical minimizer) and approximation error (difference between $R^\star_{\mathcal{C}}$ and the true Bayes risk). The guarantee above quantifies how many samples suffice to push the estimation error down to ε, whatever the irreducible noise may be.

20.9 Controlling Complexity via VC Dimension

For infinite hypothesis classes, cardinality-based bounds break down: plugging $|\mathcal{H}| = \infty$ into the finite-class estimate yields no information, and even countably infinite classes defeat a naive union bound because one must sum probabilities over infinitely many "bad" hypotheses. The *Vapnik–Chervonenkis (VC) dimension* provides a refined measure of capacity in this regime. Shattering captures the idea of flexibility: a set $A \subseteq \mathcal{X}$ is *shattered* by $\mathcal{C}$ if every possible labeling of A can be realized by some $c \in \mathcal{C}$. The VC dimension $\text{VC}_{\mathcal{C}}$ is the size of the largest shattered set.

Examples give intuition about how geometry restricts shattering power:

- Any two points in $\mathbb{R}^2$ can be shattered by axis-aligned rectangles.

- Intervals on the real line shatter any two points but not all triples.
- Certain triples in $\mathbb{R}^3$ are shattered by rectangles, but not every triple, placing $\mathrm{VC}_{\mathcal{C}}$ between two and three.

Finite VC dimension ensures that empirical risk minimization cannot overfit arbitrarily: the number of distinct labelings induced on any sample grows polynomially with n, so concentration inequalities still control the gap between empirical and true risk. Formally, if $\mathrm{VC}_{\mathcal{C}} < \infty$, empirical risk minimization achieves

$$\mathbb{P}\left(R(\hat{c}_n^*) - \inf_{c\in\mathcal{C}} R(c) > \varepsilon\right) \le 9n^{\mathrm{VC}_{\mathcal{C}}} \exp\left(-\frac{n\varepsilon^2}{32}\right),$$

which tends to zero as n grows. Thus finite VC dimension is both necessary and sufficient for PAC learnability in many settings.

20.10 Empirical Risk Minimization for Hyperplanes

Linear separators provide a concrete example of an infinite hypothesis class that is still learnable. Let $\mathcal{C}$ be the set of all hyperplanes in $\mathbb{R}^d$, which is uncountable. However, on a fixed sample of size n, only finitely many distinct labelings are possible. A *fingering* argument makes this explicit: choose any d sample points in general position (which holds with probability one under a density), and the hyperplane passing through them defines two classifiers depending on which side is labeled positive. There are at most $2\binom{n}{d}$ such classifiers, so the effective size of the class on the sample is polynomial in n for fixed d.

Moreover, for any linear classifier c there exists a hyperplane through d sample points whose empirical error differs by at most d/n. The intuition is that two hyperplanes can agree on all but the points lying on the separating plane; those are at most d points. This reduces ERM for hyperplanes to ERM over a finite class of size $2\binom{n}{d}$, so uniform convergence still holds. In particular, for $n \ge d$ and $\varepsilon \ge 2d/n$, one obtains bounds of the form

$$\mathbb{P}\left(R(\hat{c}) > \inf_{c\in\mathcal{C}} R(c) + \varepsilon\right) \le 2\binom{n}{d} \exp\left(-\frac{n\varepsilon^2}{2}\right),$$

illustrating a polynomial prefactor in n^d and an exponential decay in $n\varepsilon^2$. For fixed dimension d, the deviation probability still vanishes rapidly as n grows, confirming learnability without distributional assumptions.

If the best linear classifier has zero error ($R(c^\star) = 0$), the convergence rate improves: the probability of an ε-bad classifier scales like

$$\mathbb{P}\left(R(\hat{c}_n) > \varepsilon\right) \le 2\binom{n}{d} \exp\bigl(-\varepsilon(n-d)\bigr),$$

since errors can occur only on the $n-d$ points not used to define the separating hyperplane. This yields an $\exp(-n\varepsilon)$ dependence rather than $\exp(-n\varepsilon^2)$.

Note. Finding the optimal dichotomy of n labeled points in $\mathbb{R}^d$ is NP-hard in the worst case, so ERM for hyperplanes can be computationally difficult even though it is statistically learnable.

20.11 Strong and Weak Learning Perspectives

The PAC condition can be relaxed to *weak learning*: instead of demanding arbitrarily small ε, one requires $R(\hat{c}) \le \frac{1}{2} - \gamma$ for some fixed $\gamma > 0$ with high probability. Weak learners are tractable building blocks for ensemble methods such as boosting, which combine many slightly better-than-random hypotheses into a strongly consistent classifier. In contrast, strong PAC learning ensures (ε, δ) guarantees for every positive pair. Both notions rely on the same vocabulary—generalization error, sample complexity, and hypothesis class capacity—and highlight the central PAC message: learnability hinges on balancing expressive power with the ability to control error uniformly over the data-generating process.